# MQL5语言参考手册

## MetaTrader 5客户端专用

**研究MQL5和**
**解决任何任务：**

- 创建您自己的任何复杂的技术分析指标
- 使用自动交易 - 自动交易系统工作在不同金融市场
- 开发你自己的分析工具将实际与传统数字工具相结合
- 撰写解决信息交易系统的任务（交易，监测范围广，警报等）

© 2000-2025, MetaQuotes Ltd

# Content

MQL5参考 72
1. 语言基础 74
语法 75
注释 76
标识符 77
关键词 78
数据类型 80
整型 81
字符型，短整型，整型和长整型 83
字符常量 87
日期时间型 90
颜色型 91
布尔类型 92
枚举类型 93
真实型（double，float） 95
复数（complex） 101
字符串数据 102
架构和类 103
动态数组目标 129
矩阵和向量 130
类型转换 136
空型和NULL常量 141
用户定义类型 142
目标指针 152
引用，修饰符&和关键字this 156
运行式和表达式 158
表达式 159
算术运算 160
赋值运算 161
关系运算 162
布尔运算 163
逐位运算 165
其他运算 168
优先规则 172
操作符 174
复合操作符 175
表达式操作符 176
返回操作符 177
假设操作符if-else 178
假设操作符?： 179
循环操作符 while 181
切换操作符 182
循环操作符 for 184
循环操作符 do while 185
嵌入操作符 186
继续操作符 187
矩阵乘法运算符 @ 188
对象创建操作符new 190
对象删除操作符delete 191
函数 192
调用函数 194
传递函数 195
重载函数 198

© 2000-2025, MetaQuotes Ltd.

操作重载 201
外部函数描述 214
输出函数 216
事件处理函数 217
变量 227
局部变量 231
形式参数 233
静态变量 235
全局变量 236
输入变量 237
外部变量 243
变量初始化 244
可见范围和变量使用期 246
创建和删除对象 248
预处理程序 250
常量声明 (#define) 252
程序属性 (#property) 255
包括文件 (#include) 260
输入函数 (#import) 261
条件编译 (#ifdef, #ifndef, #else, #endif) 264
面向对象的程序设计 266
类型的密封和扩展 267
继承算法 269
多态性 273
重载 277
虚拟函数 278
类/结构的静态成员 282
函数模板 286
类模板 290
抽象类和纯虚函数 295
命名空间 297
2. 标准常量，列举和架构 300
图表常量 301
图表事件类型 302
图表时间表 309
图表性能 311
定位常量 322
图表陈述 323
使用图表示例 325
对象常量 383
物件类型 384
OBJ_VLINE 386
OBJ_HLINE 391
OBJ_TREND 396
OBJ_TRENDBYANGLE 403
OBJ_CYCLES 409
OBJ_ARROWED_LINE 415
OBJ_CHANNEL 421
OBJ_STDDEVCHANNEL 428
OBJ_REGRESSION 434
OBJ_PITCHFORK 440
OBJ_GANNLINE 448
OBJ_GANNFAN 455
OBJ_GANNGRID 462
OBJ_FIBO 469
OBJ_FIBOTIMES 476
OBJ_FIBOFAN 483
OBJ_FIBOARC 490

© 2000-2025, MetaQuotes Ltd.

OBJ_FIBOCHANNEL 497
OBJ_EXPANSION 505
OBJ_ELLIOTWAVE5 513
OBJ_ELLIOTWAVE3 521
OBJ_RECTANGLE 528
OBJ_TRIANGLE 534
OBJ_ELLIPSE 541
OBJ_ARROW_THUMB_UP 547
OBJ_ARROW_THUMB_DOWN 553
OBJ_ARROW_UP 559
OBJ_ARROW_DOWN 565
OBJ_ARROW_STOP 571
OBJ_ARROW_CHECK 577
OBJ_ARROW_LEFT_PRICE 583
OBJ_ARROW_RIGHT_PRICE 588
OBJ_ARROW_BUY 593
OBJ_ARROW_SELL 598
OBJ_ARROW 603
OBJ_TEXT 609
OBJ_LABEL 615
OBJ_BUTTON 623
OBJ_CHART 629
OBJ_BITMAP 636
OBJ_BITMAP_LABEL 643
OBJ_EDIT 650
OBJ_EVENT 657
OBJ_RECTANGLE_LABEL 662
对象属性 668
对象定位方法 690
图表角落 695
对象可见性 698
埃利奥特波动水平 701
江恩物件 702
网页颜色 704
Wingdings 706
**指标常量 707**
价格常数 708
平滑方式 711
指标线 712
绘画风格 714
自定义指标属性 718
指标类型 723
数据类型标识符 725
**环境状态 726**
客户端属性 727
运行MQL5程序属性 736
交易品种属性 739
账户属性 805
测试统计 811
**交易常数 815**
历史数据库属性 816
订单属性 817
仓位属性 830
交易属性 833
交易操作类型 837
交易事务类型 849
DOM交易订单 851
信号属性 852

© 2000-2025, MetaQuotes Ltd.

命名常量 854
预定义大量代替值 855
数学常量 861
数值类型常量 862
不能初始化原因代码 865
检测对象指针 867
其他常量 868
数据结构 872
数据类型结构 873
输入参量结构 874
历史数据结构 875
DOM 结构 876
交易请求结构 877
请求结构测试结果 890
交易请求结构结果 891
交易事务结构 895
当前价格结构 901
数据结构 902
错误和警告代码 911
交易服务器返回代码 912
编译器警告 915
编译错误 918
运行时间错误 928
输入/输出常量 940
文件开始标签 941
文件属性 943
内部文件位置 944
代码页使用 945
对话框 946
3. MQL5 程序 948
程序运行 949
交易权限 955
客户端事件 959
资源 962
调用输入函数 973
运行时间出错 975
测试交易策略 976
4. 预定义变量 999
_AppliedTo 1000
_Digits 1002
_Point 1003
_LastError 1004
_Period 1005
_RandomSeed 1006
_StopFlag 1007
_Symbol 1008
_UninitReason 1009
_IsX64 1010
5. 普通函数 1011
Alert 1013
CheckPointer 1018
Comment 1020
CryptEncode 1021
CryptDecode 1023
DebugBreak 1025
ExpertRemove 1026
GetPointer 1028

© 2000-2025, MetaQuotes Ltd.

GetTickCount 1032
GetTickCount64 1033
GetMicrosecondCount 1035
MessageBox 1037
PeriodSeconds 1042
Playsound 1044
Print 1047
PrintFormat 1049
ResetLastError 1054
ResourceCreate 1055
ResourceFree 1058
ResourceReadImage 1061
ResourceSave 1065
SetReturnError 1070
SetUserError 1072
Sleep 1073
TerminalClose 1075
TesterHideIndicators 1077
TesterStatistics 1079
TesterStop 1081
TesterDeposit 1083
TesterWithdrawal 1086
TranslateKey 1089
ZeroMemory 1090
6. 数组函数 1093
ArrayBsearch 1094
ArrayCopy 1098
ArrayCompare 1103
ArrayFree 1106
ArrayGetAsSeries 1115
ArrayInitialize 1118
ArrayFill 1120
ArrayIsDynamic 1121
ArrayIsSeries 1124
ArrayMaximum 1126
ArrayMinimum 1137
ArrayPrint 1148
ArrayRange 1150
ArrayResize 1151
ArrayInsert 1154
ArrayRemove 1156
ArrayReverse 1158
ArraySetAsSeries 1160
ArraySize 1162
ArraySort 1164
ArraySwap 1169
ArrayToFP16 1171
ArrayToFP8 1174
ArrayFromFP16 1177
ArrayFromFP8 1180
7. 矩阵和向量方法 1183
矩阵和向量类型 1189
枚举 1190
初始化 1195
Assign 1198
CopyIndicatorBuffer 1200
CopyRates 1202
CopyTicks 1207

© 2000-2025, MetaQuotes Ltd.

CopyTicksRange 1210
Eye 1212
Identity 1214
Ones 1216
Zeros 1217
Full 1218
Tri 1219
Init 1220
Fill 1222
Random 1223
矩阵和向量操纵 1225
HasNan 1226
Transpose 1227
TransposeConjugate 1229
TriL 1230
TriU 1231
Diag 1232
Row 1234
Col 1236
Copy 1238
Concat 1240
Compare 1242
CompareByDigits 1244
CompareEqual 1246
Flat 1249
Clip 1251
Reshape 1252
Resize 1254
Set 1256
SwapRows 1258
SwapCols 1259
Split 1260
Hsplit 1262
Vsplit 1264
ArgSort 1266
Sort 1267
运算 1269
数学运算 1271
数学函数 1272
乘积 1273
MatMul 1274
GeMM 1279
Power 1283
Dot 1286
Kron 1288
Inner 1290
Outer 1292
CorrCoef 1295
Cov 1298
Correlate 1301
Convolve 1304
变换 1307
Cholesky 1308
Eig 1309
EigVals 1312
LU 1313
LUP 1315
QR 1317


© 2000-2025, MetaQuotes Ltd.

SVD 1319
统计 1322
ArgMax 1323
ArgMin 1324
Max 1325
Min 1326
Ptp 1327
Sum 1328
Prod 1329
CumSum 1331
CumProd 1333
Percentile 1335
Quantile 1337
Median 1339
Mean 1341
Average 1342
Std 1344
Var 1346
LinearRegression 1348
特征 1351
Rows 1352
Cols 1353
Size 1354
Norm 1355
Cond 1358
Det 1361
SLogDet 1363
Rank 1364
Trace 1367
Spectrum 1368
解法 1369
Solve 1370
LstSq 1371
Inv 1372
PInv 1374
机器学习 1375
Activation 1380
Derivative 1384
Loss 1386
LossGradient 1388
RegressionMetric 1390
ConfusionMatrix 1392
ConfusionMatrixMultilabel 1394
ClassificationMetric 1396
ClassificationScore 1399
PrecisionRecall 1403
ReceiverOperatingCharacteristic 1407
OpenBLAS 1411
Singular Value Decomposition 1414
SingularValueDecompositionDC 1415
SingularValueDecompositionQR 1417
SingularValueDecompositionQRPivot 1419
SingularValueDecompositionBisect 1423
SingularValueDecompositionJacobiHigh 1426
SingularValueDecompositionJacobiLow 1431
SingularValueDecompositionBidiagDC 1434
SingularValueDecompositionBidiagBisect 1437
SingularValueDecompositionBidiagQR 1440

© 2000-2025, MetaQuotes Ltd.

Eigen Values ... 1442
General Matrices ... 1444
EigenSolver ... 1445
EigenSolverX ... 1447
EigenSolverShur ... 1451
EigenSolver2 ... 1453
EigenSolver2X ... 1456
EigenSolver2Shur ... 1461
EigenSolver2Blocked ... 1464
EigenSolver2ShurBlocked ... 1467
Symmetric Matrices ... 1470
EigenSymmetricDC ... 1472
EigenSymmetricQR ... 1474
EigenSymmetricRobust ... 1476
EigenSymmetricBisect ... 1479
EigenSymmetricDC2s ... 1482
EigenSymmetricQR2s ... 1484
EigenSymmetricRobust2s ... 1486
EigenSymmetricBisect2s ... 1489
EigenSymmetric2DC ... 1492
EigenSymmetric2QR ... 1495
EigenSymmetric2Bisect ... 1498
Tridiagonal Matrices ... 1502
EigenTridiagonalDC ... 1504
EigenTridiagonalQR ... 1506
EigenTridiagonalRobust ... 1508
EigenTridiagonalBisect ... 1511
EigenTridiagonalQL ... 1514
EigenTridiagonalDCQ ... 1515
EigenTridiagonalQRQ ... 1517
EigenTridiagonalPosDefQ ... 1519
Orthogonal Factorizations ... 1521
FactorizationQR ... 1522
FactorizationQRNonNeg ... 1524
FactorizationQRPivot ... 1526
FactorizationLQ ... 1528
FactorizationQL ... 1530
FactorizationRQ ... 1532
Linear Equations ... 1534
LinearEquationsSolution ... 1536
LinearEquationsSolutionTriangular ... 1538
LinearEquationsSolutionSy ... 1540
LinearEquationsSolutionComplexSy ... 1542
LinearEquationsSolutionSyPD ... 1543
LinearEquationsSolutionGeTrid ... 1545
LinearEquationsSolutionSyTridPD ... 1547
Factorizations ... 1549
FactorizationPLU ... 1550
FactorizationPLUGeTrid ... 1552
FactorizationLDL ... 1554
FactorizationLDLSyTridPD ... 1556
FactorizationCholesky ... 1558
FactorizationCholeskySyPS ... 1559
Dynamic Mode Decomposition ... 1561
DynamicModeDecomposition ... 1562
DynamicModeDecompositionQR ... 1568
Singular Spectrum Analysis ... 1575
SingularSpectrumAnalysisSpectrum ... 1576


© 2000-2025, MetaQuotes Ltd.

SingularSpectrumAnalysisForecast 1577
SingularSpectrumAnalysisReconstructComponents 1578
SingularSpectrumAnalysisReconstructSeries 1579
Reductions 1580
ReduceToBidiagonal 1581
ReflectBidiagonalToQP 1584
ReduceSymmetricToTridiagonal 1586
ReflectTridiagonalToQ 1588
8．函数转换 1590
CharToString 1591
CharArrayToString 1593
CharArrayToStruct 1595
StructToCharArray 1599
ColorToARGB 1602
ColorToString 1604
DoubleToString 1605
EnumToString 1606
IntegerToString 1608
ShortToString 1610
ShortArrayToString 1612
TimeToString 1614
NormalizeDouble 1617
StringToCharArray 1619
StringToColor 1621
StringToDouble 1622
StringToInteger 1623
StringToShortArray 1624
StringToTime 1626
StringFormat 1629
9．数学函数 1633
MathAbs 1635
MathArccos 1636
MathArcsin 1639
MathArctan 1642
MathArctan2 1645
MathClassify 1649
MathCeil 1651
MathCos 1653
MathExp 1656
MathFloor 1659
MathLog 1661
MathLog10 1665
MathMax 1669
MathMin 1671
MathMod 1673
MathPow 1674
MathRand 1678
MathRound 1679
MathSin 1681
MathSqrt 1684
MathSrand 1688
MathTan 1691
MathIsValidNumber 1694
MathExpm1 1695
MathLog1p 1698
MathArccosh 1702
MathArcsinh 1705
MathArctanh 1708

© 2000-2025, MetaQuotes Ltd.

MathCosh 1711
MathSinh 1714
MathTanh 1717
MathSwap 1720
10. 字符串函数 1723
StringAdd 1724
StringBufferLen 1726
StringCompare 1727
StringConcatenate 1729
StringFill 1731
StringFind 1732
StringGetCharacter 1735
StringInit 1737
StringLen 1738
StringSetLength 1739
StringReplace 1740
StringReserve 1741
StringSetChatacter 1743
StringSplit 1745
StringSubstr 1746
StringToLower 1747
StringToUpper 1748
StringTrimLeft 1749
StringTrimRight 1750
11. 日期和时间 1751
TimeCurrent 1752
TimeTradeServer 1754
TimeLocal 1756
TimeGMT 1758
TimeDaylightSaving 1760
TimeGMTOffset 1761
TimeToStruct 1762
StructToTime 1763
12. 账户信息 1765
AccountInfoDouble 1766
AccountInfoInteger 1767
AccountInfoString 1769
13. 检测 1770
GetLastError 1771
IsStopped 1772
UninitializeReason 1773
TerminalInfoInteger 1775
TerminalInfoDouble 1776
TerminalInfoString 1777
MQLInfoInteger 1778
MQLInfoString 1779
Symbol 1780
Period 1781
Digits 1782
Point 1783
14. 事件处理 1784
OnStart 1786
OnInit 1789
OnDeinit 1792
OnTick 1795
OnCalculate 1801

© 2000-2025, MetaQuotes Ltd.

OnTimer 1805
OnTrade 1808
OnTradeTransaction 1813
OnBookEvent 1819
OnChartEvent 1822
OnTester 1828
OnTesterInit 1835
OnTesterDeinit 1842
OnTesterPass 1843
15. 市场信息 1844
SymbolsTotal 1845
SymbolExist 1846
SymbolName 1848
SymbolSelect 1850
SymbolIsSynchronized 1852
SymbolInfoDouble 1853
SymbolInfoInteger 1855
SymbolInfoString 1857
SymbolInfoMarginRate 1859
SymbolInfoTick 1861
SymbolInfoSessionQuote 1862
SymbolInfoSessionTrade 1864
MarketBookAdd 1866
MarketBookRelease 1868
MarketBookGet 1870
16. 经济日历函数 1871
CalendarCountryById 1872
CalendarEventById 1874
CalendarValueById 1877
CalendarCountries 1880
CalendarEventByCountry 1882
CalendarEventByCurrency 1884
CalendarValueHistoryByEvent 1886
CalendarValueHistory 1889
CalendarValueLastByEvent 1892
CalendarValueLast 1897
17. 时间序列和指标访问 1901
数组，缓冲器和时序列中索引方向 1905
组织数据存储 1908
SeriesInfoInteger 1916
Bars 1918
BarsCalculated 1920
IndicatorCreate 1922
IndicatorParameters 1924
IndicatorRelease 1926
CopyBuffer 1928
CopyRates 1932
CopySeries 1936
CopyTime 1940
CopyOpen 1943
CopyHigh 1946
CopyLow 1950
CopyClose 1953
CopyTickVolume 1956
CopyRealVolume 1960
CopySpread 1963
CopyTicks 1967

© 2000-2025, MetaQuotes Ltd.

CopyTicksRange 1973
iBars 1974
iBarShift 1975
iClose 1978
iHigh 1980
iHighest 1982
iLow 1983
iLowest 1985
iOpen 1986
iTime 1988
iTickVolume 1990
iRealVolume 1992
iVolume 1994
iSpread 1996
18．自定义交易品种 1998
CustomSymbolCreate 1999
CustomSymbolDelete 2002
CustomSymbolSetInteger 2004
CustomSymbolSetDouble 2008
CustomSymbolSetString 2012
CustomSymbolSetMarginRate 2016
CustomSymbolSetSessionQuote 2020
CustomSymbolSetSessionTrade 2025
CustomRatesDelete 2030
CustomRatesReplace 2035
CustomRatesUpdate 2040
CustomTicksAdd 2045
CustomTicksDelete 2053
CustomTicksReplace 2061
CustomBookAdd 2070
19．图表操作 2073
ChartApplyTemplate 2075
ChartSaveTemplate 2078
ChartWindowFind 2083
ChartTimePriceToXY 2085
ChartXYToTimePrice 2087
ChartOpen 2089
ChartFirst 2091
ChartNext 2092
ChartClose 2093
ChartSymbol 2095
ChartPeriod 2096
ChartRedraw 2097
ChartSetDouble 2100
ChartSetInteger 2102
ChartSetString 2104
ChartGetDouble 2106
ChartGetInteger 2108
ChartGetString 2110
ChartNavigate 2112
ChartID 2115
ChartIndicatorAdd 2117
ChartIndicatorDelete 2118
ChartIndicatorGet 2121
ChartIndicatorName 2123
ChartIndicatorsTotal 2125
ChartWindowOnDropped 2127
ChartPriceOnDropped 2128

© 2000-2025, MetaQuotes Ltd.

ChartTimeOnDropped 2129
ChartXOnDropped 2130
ChartYOnDropped 2131
ChartSetSymbolPeriod 2132
ChartScreenShot 2134
20. 交易函数 2137
OrderCalcMargin 2139
OrderCalcProfit 2143
OrderCheck 2146
OrderSend 2150
OrderSendAsync 2155
PositionsTotal 2165
PositionGetSymbol 2166
PositionSelect 2168
PositionSelectByTicket 2170
PositionGetDouble 2172
PositionGetInteger 2175
PositionGetString 2177
PositionGetTicket 2180
OrdersTotal 2182
OrderGetTicket 2183
OrderSelect 2185
OrderGetDouble 2188
OrderGetInteger 2192
OrderGetString 2198
HistorySelect 2201
HistorySelectByPosition 2203
HistoryOrderSelect 2207
HistoryOrdersTotal 2210
HistoryOrderGetTicket 2211
HistoryOrderGetDouble 2213
HistoryOrderGetInteger 2217
HistoryOrderGetString 2220
HistoryDealSelect 2223
HistoryDealsTotal 2226
HistoryDealGetTicket 2227
HistoryDealGetDouble 2230
HistoryDealGetInteger 2235
HistoryDealGetString 2238
21. 交易信号 2242
SignalBaseGetDouble 2243
SignalBaseGetInteger 2244
SignalBaseGetString 2245
SignalBaseSelect 2246
SignalBaseTotal 2247
SignalInfoGetDouble 2248
SignalInfoGetInteger 2249
SignalInfoGetString 2250
SignalInfoSetDouble 2251
SignalInfoSetInteger 2252
SignalSubscribe 2253
SignalUnsubscribe 2254
22. 网络函数 2255
SocketCreate 2257
SocketClose 2260
SocketConnect 2263
SocketIsConnected 2267

© 2000-2025, MetaQuotes Ltd.

SocketIsReadable 2271
SocketIsWritable 2274
SocketTimeouts 2278
SocketRead 2280
SocketSend 2284
SocketTlsHandshake 2288
SocketTlsCertificate 2293
SocketTlsRead 2297
SocketTlsReadAvailable 2301
SocketTlsSend 2306
WebRequest 2310
SendFTP 2313
SendMail 2315
SendNotification 2317
23. 程序端全局变量 2319
GlobalVariableCheck 2320
GlobalVariableTime 2321
GlobalVariableDel 2322
GlobalVariableGet 2323
GlobalVariableName 2324
GlobalVariableSet 2325
GlobalVariablesFlush 2326
GlobalVariableTemp 2327
GlobalVariableSetOnCondition 2328
GlobalVariablesDeleteAll 2329
GlobalVariablesTotal 2330
24. 文件函数 2331
FileSelectDialog 2333
FileFindFirst 2335
FileFindNext 2337
FileFindClose 2339
FileIsExist 2340
FileOpen 2343
FileClose 2345
FileCopy 2346
FileDelete 2349
FileMove 2352
FileFlush 2354
FileGetInteger 2356
FileIsEnding 2359
FileIsLineEnding 2360
FileReadArray 2366
FileReadBool 2368
FileReadDatetime 2372
FileReadDouble 2376
FileReadFloat 2379
FileReadInteger 2383
FileReadLong 2387
FileReadNumber 2390
FileReadString 2396
FileReadStruct 2398
FileSeek 2402
FileSize 2405
FileTell 2408
FileWrite 2411
FileWriteArray 2414
FileWriteDouble 2417
FileWriteFloat 2420

© 2000-2025, MetaQuotes Ltd.

FileWriteInteger 2423
FileWriteLong 2426
FileWriteString 2429
FileWriteStruct 2432
FileLoad 2435
FileSave 2437
FolderCreate 2439
FolderDelete 2442
FolderClean 2445
25. 自定义指标 2448
指标类型示例 2451
DRAW_NONE 2459
DRAW_LINE 2462
DRAW_SECTION 2466
DRAW_HISTOGRAM 2470
DRAW_HISTOGRAM2 2474
DRAW_ARROW 2478
DRAW_ZIGZAG 2482
DRAW_FILLING 2487
DRAW_BARS 2492
DRAW_CANDLES 2498
DRAW_COLOR_LINE 2504
DRAW_COLOR_SECTION 2509
DRAW_COLOR_HISTOGRAM 2514
DRAW_COLOR_HISTOGRAM2 2519
DRAW_COLOR_ARROW 2524
DRAW_COLOR_ZIGZAG 2530
DRAW_COLOR_BARS 2535
DRAW_COLOR_CANDLES 2541
指标属性和函数间的连接 2547
SetIndexBuffer 2549
IndicatorSetDouble 2552
IndicatorSetInteger 2556
IndicatorSetString 2560
PlotIndexSetDouble 2563
PlotIndexSetInteger 2564
PlotIndexSetString 2568
PlotIndexGetInteger 2569
26. 物件函数 2572
ObjectCreate 2573
ObjectName 2577
ObjectDelete 2578
ObjectsDeleteAll 2579
ObjectFind 2580
ObjectGetTimeByValue 2581
ObjectGetValueByTime 2582
ObjectMove 2583
ObjectsTotal 2584
ObjectSetDouble 2585
ObjectSetInteger 2588
ObjectSetString 2592
ObjectGetDouble 2594
ObjectGetInteger 2595
ObjectGetString 2596
TextSetFont 2598
TextOut 2600
TextGetSize 2604

© 2000-2025, MetaQuotes Ltd.

27．技术指标 2605
iAC 2607
iAD 2612
iADX 2617
iADXWilder 2622
iAlligator 2627
iAMA 2634
iAO 2639
iATR 2644
iBearsPower 2649
iBands 2654
iBullsPower 2660
iCCI 2665
iChaikin 2670
iCustom 2675
iDEMA 2678
iDeMarker 2683
iEnvelopes 2688
iForce 2694
iFractals 2699
iFrAMA 2704
iGator 2709
iIchimoku 2716
iBWMFI 2723
iMomentum 2728
iMFI 2733
iMA 2738
iOsMA 2743
iMACD 2748
iOBV 2754
iSAR 2759
iRSI 2764
iRVI 2769
iStdDev 2774
iStochastic 2779
iTEMA 2785
iTriX 2790
iWPR 2795
iVIDyA 2800
iVolumes 2805
28．处理优化结果 2810
FrameFirst 2812
FrameFilter 2813
FrameNext 2814
FrameInputs 2815
FrameAdd 2816
ParameterGetRange 2817
ParameterSetRange 2820
29．工作事件 2822
EventSetMillisecondTimer 2823
EventSetTimer 2824
EventKillTimer 2825
EventChartCustom 2826
30．使用OpenCL工作 2834
CLHandleType 2836
CLGetInfoInteger 2837
CLGetInfoString 2840

© 2000-2025, MetaQuotes Ltd.

CLContextCreate 2843
CLContextFree 2844
CLGetDeviceInfo 2845
CLProgramCreate 2850
CLProgramFree 2854
CLKernelCreate 2855
CLKernelFree 2856
CLSetKernelArg 2857
CLSetKernelArgMem 2858
CLSetKernelArgMemLocal 2859
CLBufferCreate 2860
CLBufferFree 2861
CLBufferWrite 2862
CLBufferRead 2867
CLExecute 2871
CLExecutionStatus 2873
31．使用数据库 2874
DatabaseOpen 2877
DatabaseClose 2879
DatabaseImport 2880
DatabaseExport 2883
DatabasePrint 2889
DatabaseTableExists 2894
DatabaseExecute 2895
DatabasePrepare 2907
DatabaseReset 2916
DatabaseBind 2922
DatabaseBindArray 2927
DatabaseRead 2932
DatabaseReadBind 2933
DatabaseFinalize 2937
DatabaseTransactionBegin 2938
DatabaseTransactionCommit 2943
DatabaseTransactionRollback 2944
DatabaseColumnsCount 2945
DatabaseColumnName 2946
DatabaseColumnType 2947
DatabaseColumnSize 2948
DatabaseColumnText 2949
DatabaseColumnInteger 2950
DatabaseColumnLong 2951
DatabaseColumnDouble 2952
DatabaseColumnBlob 2953
32．使用DirectX 2954
DXContextCreate 2956
DXContextSetSize 2957
DXContextGetSize 2958
DXContextClearColors 2959
DXContextClearDepth 2960
DXContextGetColors 2961
DXContextGetDepth 2962
DXBufferCreate 2963
DXTextureCreate 2964
DXInputCreate 2970
DXInputSet 2971
DXShaderCreate 2972
DXShaderSetLayout 2973
DXShaderInputsSet 2974

© 2000-2025, MetaQuotes Ltd.

DXShaderTexturesSet 2975
DXDraw 2976
DXDrawIndexed 2977
DXPrimiveTopologySet 2978
DXBufferSet 2979
DXShaderSet 2980
DXHandleType 2981
DXRelease 2982
33. MetaTrader Python模块 2983
initialize 2988
login 2990
shutdown 2993
version 2994
last_error 2996
account_info 2998
terminal_info 3001
symbols_total 3004
symbols_get 3005
symbol_info 3008
symbol_info_tick 3012
symbol_select 3014
market_book_add 3018
market_book_get 3019
market_book_release 3022
copy_rates_from 3023
copy_rates_from_pos 3027
copy_rates_range 3030
copy_ticks_from 3033
copy_ticks_range 3036
terminal_info 3039
orders_total 3042
orders_get 3043
order_calc_margin 3046
order_calc_profit 3049
order_check 3052
order_send 3056
positions_total 3061
positions_get 3062
history_deals_total 3065
history_deals_get 3067
history_orders_total 3070
history_orders_get 3072
34. ONNX模型 3075
ONNX支持 3076
格式转换 3077
自动数据类型转换 3078
创建模型 3081
运行模型 3088
运行策略测试 3094
OnnxCreate 3098
OnnxCreateFromBuffer 3099
OnnxRelease 3100
OnnxRun 3101
OnnxGetInputCount 3104
OnnxGetOutputCount 3105
OnnxGetInputName 3106
OnnxGetOutputName 3107
OnnxGetInputTypeInfo 3108

© 2000-2025, MetaQuotes Ltd.

OnnxGetOutputTypeInfo 3109
OnnxSetInputShape 3110
OnnxSetOutputShape 3111
数据结构 3112
35 . 标准程序库 3116
数学 3117
统计 3118
统计特性 3121
MathMean 3122
MathVariance 3123
MathSkewness 3124
MathKurtosis 3125
MathMoments 3126
MathMedian 3127
MathStandardDeviation 3128
MathAverageDeviation 3129
正态分布 3130
MathProbabilityDensityNormal 3134
MathCumulativeDistributionNormal 3136
MathQuantileNormal 3138
MathRandomNormal 3140
MathMomentsNormal 3141
对数正态分布 3142
MathProbabilityDensityLognormal 3146
MathCumulativeDistributionLognormal 3148
MathQuantileLognormal 3150
MathRandomLognormal 3152
MathMomentsLognormal 3153
Beta分布 3154
MathProbabilityDensityBeta 3158
MathCumulativeDistributionBeta 3160
MathQuantileBeta 3162
MathRandomBeta 3164
MathMomentsBeta 3165
非中心beta分布 3166
MathProbabilityDensityNoncentralBeta 3170
MathCumulativeDistributionNoncentralBeta 3172
MathQuantileNoncentralBeta 3174
MathRandomNoncentralBeta 3176
MathMomentsNoncentralBeta 3177
伽玛分布 3178
MathProbabilityDensityGamma 3182
MathCumulativeDistributionGamma 3184
MathQuantileGamma 3186
MathRandomGamma 3188
MathMomentsGamma 3189
卡方分布 3190
MathProbabilityDensityChiSquare 3194
MathCumulativeDistributionChiSquare 3196
MathQuantileChiSquare 3198
MathRandomChiSquare 3200
MathMomentsChiSquare 3201
非中心卡方分布 3202
MathProbabilityDensityNoncentralChiSquare 3206
MathCumulativeDistributionNoncentralChiSquare 3208
MathQuantileNoncentralChiSquare 3210
MathRandomNoncentralChiSquare 3212
MathMomentsNoncentralChiSquare 3213

© 2000-2025, MetaQuotes Ltd.

指数分布 3214
MathProbabilityDensityExponential 3218
MathCumulativeDistributionExponential 3220
MathQuantileExponential 3222
MathRandomExponential 3224
MathMomentsExponential 3225
F-分布 3226
MathProbabilityDensityF 3230
MathCumulativeDistributionF 3232
MathQuantileF 3234
MathRandomF 3236
MathMomentsF 3237
非中心F-分布 3238
MathProbabilityDensityNoncentralF 3242
MathCumulativeDistributionNoncentralF 3244
MathQuantileNoncentralF 3246
MathRandomNoncentralF 3248
MathMomentsNoncentralF 3249
t-分布 3250
MathProbabilityDensityT 3254
MathCumulativeDistributionT 3256
MathQuantileT 3258
MathRandomT 3260
MathMomentsT 3261
非中心t-分布 3262
MathProbabilityDensityNoncentralT 3266
MathCumulativeDistributionNoncentralT 3268
MathQuantileNoncentralT 3270
MathRandomNoncentralT 3272
MathMomentsNoncentralT 3273
逻辑斯谛分布 3274
MathProbabilityDensityLogistic 3278
MathCumulativeDistributionLogistic 3280
MathQuantileLogistic 3282
MathRandomLogistic 3284
MathMomentsLogistic 3285
柯西分布 3286
MathProbabilityDensityCauchy 3290
MathCumulativeDistributionCauchy 3292
MathQuantileCauchy 3294
MathRandomCauchy 3296
MathMomentsCauchy 3297
均匀分布 3298
MathProbabilityDensityUniform 3302
MathCumulativeDistributionUniform 3304
MathQuantileUniform 3306
MathRandomUniform 3308
MathMomentsUniform 3309
威布尔分布 3310
MathProbabilityDensityWeibull 3314
MathCumulativeDistributionWeibull 3316
MathQuantileWeibull 3318
MathRandomWeibull 3320
MathMomentsWeibull 3321
二项式分布 3322
MathProbabilityDensityBinomial 3325
MathCumulativeDistributionBinomial 3327
MathQuantileBinomial 3329

© 2000-2025, MetaQuotes Ltd.

MathRandomBinomial 3331
MathMomentsBinomial 3332
负二项式分布 3333
MathProbabilityDensityNegativeBinomial 3336
MathCumulativeDistributionNegativeBinomial 3338
MathQuantileNegativeBinomial 3340
MathRandomNegativeBinomial 3342
MathMomentsNegativeBinomial 3343
几何分布 3344
MathProbabilityDensityGeometric 3348
MathCumulativeDistributionGeometric 3350
MathQuantileGeometric 3352
MathRandomGeometric 3354
MathMomentsGeometric 3355
超几何分布 3356
MathProbabilityDensityHypergeometric 3360
MathCumulativeDistributionHypergeometric 3362
MathQuantileHypergeometric 3364
MathRandomHypergeometric 3366
MathMomentsHypergeometric 3367
泊松分布 3368
MathProbabilityDensityPoisson 3372
MathCumulativeDistributionPoisson 3374
MathQuantilePoisson 3376
MathRandomPoisson 3378
MathMomentsPoisson 3379
子函数 3380
MathRandomNonZero 3384
MathMoments 3385
MathPowInt 3386
MathFactorial 3387
MathTrunc 3388
MathRound 3389
MathArctan2 3391
MathGamma 3393
MathGammaLog 3394
MathBeta 3395
MathBetaLog 3396
MathBetaIncomplete 3397
MathGammaIncomplete 3398
MathBinomialCoefficient 3399
MathBinomialCoefficientLog 3400
MathHypergeometric2F2 3401
MathSequence 3402
MathSequenceByCount 3403
MathReplicate 3404
MathReverse 3405
MathIdentical 3406
MathUnique 3407
MathQuickSortAscending 3408
MathQuickSortDescending 3409
MathQuickSort 3410
MathOrder 3411
MathBitwiseNot 3412
MathBitwiseAnd 3413
MathBitwiseOr 3414
MathBitwiseXor 3415
MathBitwiseShiftL 3416

© 2000-2025, MetaQuotes Ltd.

MathBitwiseShiftR 3417
MathCumulativeSum 3418
MathCumulativeProduct 3419
MathCumulativeMin 3420
MathCumulativeMax 3421
MathSin 3422
MathCos 3423
MathTan 3424
MathArcsin 3425
MathArccos 3426
MathArctan 3427
MathSinPi 3428
MathCosPi 3429
MathTanPi 3430
MathAbs 3431
MathCeil 3432
MathFloor 3433
MathSqrt 3434
MathExp 3435
MathPow 3436
MathLog 3437
MathLog2 3438
MathLog10 3439
MathLog1p 3440
MathDifference 3441
MathSample 3443
MathTukeySummary 3445
MathRange 3446
MathMin 3447
MathMax 3448
MathSum 3449
MathProduct 3450
MathStandardDeviation 3451
MathAverageDeviation 3452
MathMedian 3453
MathMean 3454
MathVariance 3455
MathSkewness 3456
MathKurtosis 3457
MathExpm1 3458
MathSinh 3459
MathCosh 3460
MathTanh 3461
MathArcsinh 3462
MathArccosh 3463
MathArctanh 3464
MathSignif 3465
MathRank 3467
MathCorrelationPearson 3468
MathCorrelationSpearman 3469
MathCorrelationKendall 3470
MathQuantile 3471
MathProbabilityDensityEmpirical 3472
MathCumulativeDistributionEmpirical 3473
模糊逻辑 3474
Membership functions 3475
CConstantMembershipFunction 3476
GetValue 3478

© 2000-2025, MetaQuotes Ltd.

CCompositeMembershipFunction 3479
CompositionType 3481
MembershipFunctions 3481
GetValue 3481
CDifferencTwoSigmoidalMembershipFunction 3483
A1 3485
A2 3485
C1 3486
C2 3486
GetValue 3486
CGeneralizedBellShapedMembershipFunction 3488
A 3490
B 3490
C 3491
GetValue 3491
CNormalCombinationMembershipFunction 3492
B1 3494
B2 3494
Sigma1 3495
Sigma2 3495
GetValue 3496
CNormalMembershipFunction 3497
B 3499
Sigma 3499
GetValue 3499
CP_ShapedMembershipFunction 3501
A 3503
B 3503
C 3504
D 3504
GetValue 3504
CProductTwoSigmoidalMembershipFunctions 3506
A1 3508
A2 3508
C1 3509
C2 3509
GetValue 3509
CS_ShapedMembershipFunction 3511
A 3513
B 3513
GetValue 3513
CSigmoidalMembershipFunction 3515
A 3517
C 3517
GetValue 3517
CTrapezoidMembershipFunction 3519
X1 3521
X2 3521
X3 3522
X4 3522
GetValue 3522
CTriangularMembershipFunction 3524
X1 3526
X2 3526
X3 3527
ToNormalMF 3527
GetValue 3527
CZ_ShapedMembershipFunction 3528

© 2000-2025, MetaQuotes Ltd.

A 3530
B 3530
GetValue 3530
IMembershipFunction 3532
GetValue 3532
Fuzzy systems rules 3533
CMamdaniFuzzyRule 3534
Conclusion 3534
Weight 3535
CSugenoFuzzyRule 3536
Conclusion 3536
CSingleCondition 3538
Not 3538
Term 3539
Var 3539
CConditions 3541
ConditionsList 3541
Not 3542
Op 3542
CGenericFuzzyRule 3543
Conclusion 3543
Condition 3544
CreateCondition 3544
Fuzzy systems variables 3546
CFuzzyVariable 3547
AddTerm 3548
GetTermByName 3548
Max 3548
Min 3549
Terms 3549
Values 3549
CSugenoVariable 3551
Functions 3551
GetFuncByName 3552
Values 3552
Fuzzy terms 3553
MembershipFunction 3554
Fuzzy systems 3555
Mamdani system 3556
AggregationMethod 3556
Calculate 3557
DefuzzificationMethod 3557
EmptyRule 3557
ImplicationMethod 3557
Output 3558
OutputByName 3558
ParseRule 3558
Rules 3558
Sugeno system 3560
Calculate 3560
CreateSugenoFunction 3561
EmptyRule 3562
Output 3562
OutputByName 3562
ParseRule 3562
Rules 3563
OpenCL 3564
BufferCreate 3566


© 2000-2025, MetaQuotes Ltd.

.....BufferFree..... 3567
.....BufferFromArray..... 3568
.....BufferRead..... 3569
.....BufferWrite..... 3570
.....Execute..... 3571
.....GetContext..... 3572
.....GetKernel..... 3573
.....GetKernelName..... 3574
.....GetProgram..... 3575
.....Initialize..... 3576
.....KernelCreate..... 3577
.....KernelFree..... 3578
.....SetArgument..... 3579
.....SetArgumentBuffer..... 3580
.....SetArgumentLocalMemory..... 3581
.....SetBuffersCount..... 3582
.....SetKernelsCount..... 3583
.....Shutdown..... 3584
.....SupportDouble..... 3585
.....基本类CObject ..... 3586
.....Prev..... 3587
.....Prev..... 3588
.....Next..... 3589
.....Next..... 3590
.....Compare..... 3591
.....Save..... 3593
.....Load..... 3595
.....Type..... 3597
.....数据采集 ..... 3598
.....CArray..... 3599
.....Step..... 3601
.....Step..... 3602
.....Total..... 3603
.....Available..... 3604
.....Max..... 3605
.....IsSorted..... 3606
.....SortMode..... 3607
.....Clear..... 3608
.....Sort..... 3609
.....Save..... 3610
.....Load..... 3611
.....CArrayChar..... 3612
.....Reserve..... 3614
.....Resize..... 3615
.....Shutdown..... 3616
.....Add..... 3617
.....AddArray..... 3618
.....AddArray..... 3619
.....Insert..... 3621
.....InsertArray..... 3622
.....InsertArray..... 3623
.....AssignArray..... 3625
.....AssignArray..... 3626
.....Update..... 3628
.....Shift..... 3629
.....Delete..... 3630
.....DeleteRange..... 3631
.....At..... 3632


© 2000-2025, MetaQuotes Ltd.

CompareArray 3634
CompareArray 3635
InsertSort 3636
Search 3637
SearchGreat 3638
SearchLess 3639
SearchGreatOrEqual 3640
SearchLessOrEqual 3641
SearchFirst 3642
SearchLast 3643
SearchLinear 3644
Save 3645
Load 3646
Type 3648
CArrayShort 3649
Reserve 3651
Resize 3652
Shutdown 3653
Add 3654
AddArray 3655
AddArray 3656
Insert 3658
InsertArray 3659
InsertArray 3660
AssignArray 3662
AssignArray 3663
Update 3665
Shift 3666
Delete 3667
DeleteRange 3668
At 3669
CompareArray 3671
CompareArray 3672
InsertSort 3673
Search 3674
SearchGreat 3675
SearchLess 3676
SearchGreatOrEqual 3677
SearchLessOrEqual 3678
SearchFirst 3679
SearchLast 3680
SearchLinear 3681
Save 3682
Load 3684
Type 3686
CArrayInt 3687
Reserve 3689
Resize 3690
Shutdown 3691
Add 3692
AddArray 3693
AddArray 3694
Insert 3696
InsertArray 3697
InsertArray 3698
AssignArray 3700
AssignArray 3701
Update 3703


© 2000-2025, MetaQuotes Ltd.

.....Shift..... 3704
.....Delete..... 3705
.....DeleteRange..... 3706
.....At..... 3707
.....CompareArray..... 3709
.....CompareArray..... 3710
.....InsertSort..... 3711
.....Search..... 3712
.....SearchGreat..... 3713
.....SearchLess..... 3714
.....SearchGreatOrEqual..... 3715
.....SearchLessOrEqual..... 3716
.....SearchFirst..... 3717
.....SearchLast..... 3718
.....SearchLinear..... 3719
.....Save..... 3720
.....Load..... 3722
.....Type..... 3724
.....CArrayLong..... 3725
.....Reserve..... 3727
.....Resize..... 3728
.....Shutdown..... 3729
.....Add..... 3730
.....AddArray..... 3731
.....AddArray..... 3732
.....Insert..... 3734
.....InsertArray..... 3735
.....InsertArray..... 3736
.....AssignArray..... 3738
.....AssignArray..... 3739
.....Update..... 3741
.....Shift..... 3742
.....Delete..... 3743
.....DeleteRange..... 3744
.....At..... 3745
.....CompareArray..... 3747
.....CompareArray..... 3748
.....InsertSort..... 3749
.....Search..... 3750
.....SearchGreat..... 3751
.....SearchLess..... 3752
.....SearchGreatOrEqual..... 3753
.....SearchLessOrEqual..... 3754
.....SearchFirst..... 3755
.....SearchLast..... 3756
.....SearchLinear..... 3757
.....Save..... 3758
.....Load..... 3760
.....Type..... 3762
.....CArrayFloat..... 3763
.....Delta..... 3765
.....Reserve..... 3766
.....Resize..... 3767
.....Shutdown..... 3768
.....Add..... 3769
.....AddArray..... 3770
.....AddArray..... 3771
.....Insert..... 3773


© 2000-2025, MetaQuotes Ltd.

InsertArray 3774
InsertArray 3775
AssignArray 3777
AssignArray 3778
Update 3780
Shift 3781
Delete 3782
DeleteRange 3783
At 3784
CompareArray 3786
CompareArray 3787
InsertSort 3788
Search 3789
SearchGreat 3790
SearchLess 3791
SearchGreatOrEqual 3792
SearchLessOrEqual 3793
SearchFirst 3794
SearchLast 3795
SearchLinear 3796
Save 3797
Load 3799
Type 3801
CArrayDouble 3802
Delta 3805
Reserve 3806
Resize 3807
Shutdown 3808
Add 3809
AddArray 3810
AddArray 3811
Insert 3813
InsertArray 3814
InsertArray 3815
AssignArray 3817
AssignArray 3818
Update 3820
Shift 3821
Delete 3822
DeleteRange 3823
At 3824
CompareArray 3826
CompareArray 3827
Minimum 3828
Maximum 3829
InsertSort 3830
Search 3831
SearchGreat 3832
SearchLess 3833
SearchGreatOrEqual 3834
SearchLessOrEqual 3835
SearchFirst 3836
SearchLast 3837
SearchLinear 3838
Save 3839
Load 3841
Type 3843
CArrayString 3844


© 2000-2025, MetaQuotes Ltd.

.....Reserve..... 3846
.....Resize..... 3847
.....Shutdown..... 3848
.....Add..... 3849
.....AddArray..... 3850
.....AddArray..... 3851
.....Insert..... 3853
.....InsertArray..... 3854
.....InsertArray..... 3855
.....AssignArray..... 3857
.....AssignArray..... 3858
.....Update..... 3860
.....Shift..... 3861
.....Delete..... 3862
.....DeleteRange..... 3863
.....At..... 3864
.....CompareArray..... 3866
.....CompareArray..... 3867
.....InsertSort..... 3868
.....Search..... 3869
.....SearchGreat..... 3870
.....SearchLess..... 3871
.....SearchGreatOrEqual..... 3872
.....SearchLessOrEqual..... 3873
.....SearchFirst..... 3874
.....SearchLast..... 3875
.....SearchLinear..... 3876
.....Save..... 3877
.....Load..... 3879
.....Type..... 3881
.....CArrayObj..... 3882
.....FreeMode..... 3887
.....FreeMode..... 3888
.....Reserve..... 3890
.....Resize..... 3891
.....Clear..... 3892
.....Shutdown..... 3893
.....CreateElement..... 3894
.....Add..... 3896
.....AddArray..... 3897
.....Insert..... 3900
.....InsertArray..... 3902
.....AssignArray..... 3904
.....Update..... 3906
.....Shift..... 3907
.....Detach..... 3908
.....Delete..... 3909
.....DeleteRange..... 3910
.....At..... 3911
.....CompareArray..... 3912
.....InsertSort..... 3913
.....Search..... 3914
.....SearchGreat..... 3915
.....SearchLess..... 3916
.....SearchGreatOrEqual..... 3917
.....SearchLessOrEqual..... 3918
.....SearchFirst..... 3919
.....SearchLast..... 3920

© 2000-2025, MetaQuotes Ltd.

.....Save..... 3921
.....Load..... 3922
.....Type..... 3924
.....CList..... 3925
.....FreeMode..... 3927
.....FreeMode..... 3928
.....Total..... 3929
.....IsSorted..... 3930
.....SortMode..... 3931
.....CreateElement..... 3932
.....Add..... 3933
.....Insert..... 3934
.....DetachCurrent..... 3936
.....DeleteCurrent..... 3937
.....Delete..... 3938
.....Clear..... 3939
.....IndexOf..... 3940
.....GetNodeAtIndex..... 3941
.....GetFirstNode..... 3942
.....GetPrevNode..... 3943
.....GetCurrentNode..... 3944
.....GetNextNode..... 3945
.....GetLastNode..... 3946
.....Sort..... 3947
.....MoveToIndex..... 3948
.....Exchange..... 3949
.....CompareList..... 3950
.....Search..... 3951
.....Save..... 3952
.....Load..... 3954
.....Type..... 3956
.....CTreeNode..... 3957
.....Owner..... 3962
.....Left..... 3963
.....Right..... 3964
.....Balance..... 3965
.....BalanceL..... 3966
.....BalanceR..... 3967
.....CreateSample..... 3968
.....RefreshBalance..... 3969
.....GetNext..... 3970
.....SaveNode..... 3971
.....LoadNode..... 3972
.....Type..... 3973
.....CTree..... 3974
.....Root..... 3979
.....CreateElement..... 3980
.....Insert..... 3981
.....Detach..... 3982
.....Delete..... 3983
.....Clear..... 3984
.....Find..... 3985
.....Save..... 3986
.....Load..... 3987
.....Type..... 3988
.....通用数据集 ..... 3989
.....ICollection<T>..... 3991
.....Add..... 3992


© 2000-2025, MetaQuotes Ltd.

Count 3993
Contains 3994
CopyTo 3995
Clear 3996
Remove 3997
IEqualityComparable<T> 3998
Equals 3999
HashCode 4000
IComparable<T> 4001
Compare 4002
IComparer<T> 4003
Compare 4004
IEqualityComparer<T> 4005
Equals 4006
HashCode 4007
IList<T> 4008
TryGetValue 4009
TrySetValue 4010
Insert 4011
IndexOf 4012
LastIndexOf 4013
RemoveAt 4014
IMap<TKey,TValue> 4015
Add 4016
Contains 4017
Remove 4018
TryGetValue 4019
TrySetValue 4020
CopyTo 4021
ISet<T> 4022
ExceptWith 4023
IntersectWith 4024
SymmetricExceptWith 4025
UnionWith 4026
IsProperSubsetOf 4027
IsProperSupersetOf 4028
IsSubsetOf 4029
IsSupersetOf 4030
Overlaps 4031
SetEquals 4032
CDefaultComparer<T> 4033
Compare 4034
CDefaultEqualityComparer<T> 4035
Equals 4036
HashCode 4037
CRedBlackTreeNode<T> 4038
Value 4039
Parent 4040
Left 4041
Right 4042
Color 4043
IsLeaf 4044
CreateEmptyNode 4045
CLinkedListNode<T> 4046
List 4047
Next 4048
Previous 4049
Value 4050


© 2000-2025, MetaQuotes Ltd.

CKeyValuePair<TKey,TValue> 4051
Key 4052
Value 4053
Clone 4054
Compare 4055
Equals 4056
HashCode 4057
CArrayList<T> 4058
Capacity 4060
Count 4061
Contains 4062
TrimExcess 4063
TryGetValue 4064
TrySetValue 4065
Add 4066
AddRange 4067
Insert 4068
InsertRange 4069
CopyTo 4070
BinarySearch 4071
IndexOf 4072
LastIndexOf 4073
Clear 4074
Remove 4075
RemoveAt 4076
RemoveRange 4077
Reverse 4078
Sort 4079
CHashMap<TKey,TValue> 4080
Add 4081
Count 4082
Comparer 4083
Contains 4084
ContainsKey 4085
ContainsValue 4086
CopyTo 4087
Clear 4088
Remove 4089
TryGetValue 4090
TrySetValue 4091
CHashSet<T> 4092
Add 4094
Count 4095
Contains 4096
Comparer 4097
TrimExcess 4098
CopyTo 4099
Clear 4100
Remove 4101
ExceptWith 4102
IntersectWith 4103
SymmetricExceptWith 4104
UnionWith 4105
IsProperSubsetOf 4106
IsProperSupersetOf 4107
IsSubsetOf 4108
IsSupersetOf 4109
Overlaps 4110

© 2000-2025, MetaQuotes Ltd.

SetEquals 4111
CLinkedList<T> 4112
Add 4114
AddAfter 4115
AddBefore 4116
AddFirst 4117
AddLast 4118
Count 4119
Head 4120
First 4121
Last 4122
Contains 4123
CopyTo 4124
Clear 4125
Remove 4126
RemoveFirst 4127
RemoveLast 4128
Find 4129
FindLast 4130
CQueue<T> 4131
Add 4132
Enqueue 4133
Count 4134
Contains 4135
TrimExcess 4136
CopyTo 4137
Clear 4138
Remove 4139
Dequeue 4140
Peek 4141
CRedBlackTree<T> 4142
Add 4143
Count 4144
Root 4145
Contains 4146
Comparer 4147
TryGetMin 4148
TryGetMax 4149
CopyTo 4150
Clear 4151
Remove 4152
RemoveMin 4153
RemoveMax 4154
Find 4155
FindMin 4156
FindMax 4157
CSortedMap<TKey, TValue> 4158
Add 4159
Count 4160
Comparer 4161
Contains 4162
ContainsKey 4163
ContainsValue 4164
CopyTo 4165
Clear 4166
Remove 4167
TryGetValue 4168
TrySetValue 4169


© 2000-2025, MetaQuotes Ltd.

CSortedSet<T> 4170
Add 4172
Count 4173
Contains 4174
Comparer 4175
TryGetMin 4176
TryGetMax 4177
CopyTo 4178
Clear 4179
Remove 4180
ExceptWith 4181
IntersectWith 4182
SymmetricExceptWith 4183
UnionWith 4184
IsProperSubsetOf 4185
IsProperSupersetOf 4186
IsSubsetOf 4187
IsSupersetOf 4188
Overlaps 4189
SetEquals 4190
GetViewBetween 4191
GetReverse 4192
CStack<T> 4193
Add 4194
Count 4195
Contains 4196
TrimExcess 4197
CopyTo 4198
Clear 4199
Remove 4200
Push 4201
Peek 4202
Pop 4203
ArrayBinarySearch<T> 4204
ArrayIndexOf<T> 4205
ArrayLastIndexOf<T> 4206
ArrayReverse<T> 4207
Compare 4208
Equals<T> 4211
GetHashCode 4212
文件 4214
CFile 4215
Handle 4217
Filename 4218
Flags 4219
SetUnicode 4220
SetCommon 4221
Open 4222
Close 4223
Delete 4224
IsExist 4225
Copy 4226
Move 4227
Size 4228
Tell 4229
Seek 4230
Flush 4231
IsEnding 4232

© 2000-2025, MetaQuotes Ltd.

IsLineEnding 4233
FolderCreate 4234
FolderDelete 4235
FolderClean 4236
FileFindFirst 4237
FileFindNext 4238
FileFindClose 4239
CFileBin 4240
Open 4242
WriteChar 4243
WriteShort 4244
WriteInteger 4245
WriteLong 4246
WriteFloat 4247
WriteDouble 4248
WriteString 4249
WriteCharArray 4250
WriteShortArray 4251
WriteIntegerArray 4252
WriteLongArray 4253
WriteFloatArray 4254
WriteDoubleArray 4255
WriteObject 4256
ReadChar 4257
ReadShort 4258
ReadInteger 4259
ReadLong 4260
ReadFloat 4261
ReadDouble 4262
ReadString 4263
ReadCharArray 4264
ReadShortArray 4265
ReadIntegerArray 4266
ReadLongArray 4267
ReadFloatArray 4268
ReadDoubleArray 4269
ReadObject 4270
CFileTxt 4271
Open 4272
WriteString 4273
ReadString 4274
**字符串 4275**
CString 4276
Str 4278
Len 4279
Copy 4280
Fill 4281
Assign 4282
Append 4283
Insert 4284
Compare 4285
CompareNoCase 4286
Left 4287
Right 4288
Mid 4289
Trim 4290
TrimLeft 4291
TrimRight 4292


© 2000-2025, MetaQuotes Ltd.

.....Clear..... 4293
.....ToUpper..... 4294
.....ToLower..... 4295
.....Reverse..... 4296
.....Find..... 4297
.....FindRev..... 4298
.....Remove..... 4299
.....Replace..... 4300
..... 图形对象 .....4301
.....CChartObject..... 4302
.....ChartId..... 4305
.....Window..... 4306
.....Name..... 4307
.....NumPoints..... 4308
.....Attach..... 4309
.....SetPoint..... 4310
.....Delete..... 4311
.....Detach..... 4312
.....ShiftObject..... 4313
.....ShiftPoint..... 4314
.....Time..... 4315
.....Price..... 4317
.....Color..... 4319
.....Style..... 4320
.....Width..... 4321
.....Background..... 4322
.....Selected..... 4323
.....Selectable..... 4324
.....Description..... 4325
.....Tooltip..... 4326
.....Timeframes..... 4327
.....Z_Order..... 4328
.....CreateTime..... 4329
.....LevelsCount..... 4330
.....LevelColor..... 4331
.....LevelStyle..... 4333
.....LevelWidth..... 4335
.....LevelValue..... 4337
.....LevelDescription..... 4339
.....GetInteger..... 4341
.....SetInteger..... 4343
.....GetDouble..... 4345
.....SetDouble..... 4347
.....GetString..... 4349
.....SetString..... 4351
.....Save..... 4353
.....Load..... 4354
.....Type..... 4355
.....Objects Lines..... 4356
.....CChartObjectVLine..... 4357
.....Create ..... 4358
.....Type ..... 4359
.....CChartObjectHLine..... 4360
.....Create ..... 4361
.....Type ..... 4362
.....CChartObjectTrend..... 4363
.....Create ..... 4365
.....RayLeft ..... 4366


© 2000-2025, MetaQuotes Ltd.

.....RayRight ..... 4367
.....Save ..... 4368
.....Load ..... 4369
.....Type ..... 4370
.....CChartObjectTrendByAngle..... 4371
.....Create ..... 4373
.....Angle ..... 4374
.....Type ..... 4375
.....CChartObjectCycles..... 4376
.....Create ..... 4377
.....Type ..... 4378
.....Objects Channels..... 4379
.....CChartObjectChannel..... 4380
.....Create ..... 4381
.....Type ..... 4382
.....CChartObjectRegression..... 4383
.....Create ..... 4384
.....Type ..... 4385
.....CChartObjectStdDevChannel..... 4386
.....Create ..... 4388
.....Deviations ..... 4389
.....Save ..... 4390
.....Load ..... 4391
.....Type ..... 4392
.....CChartObjectPitchfork..... 4393
.....Create ..... 4394
.....Type ..... 4395
.....Gann Tools..... 4396
.....CChartObjectGannLine..... 4397
.....Create ..... 4399
.....PipsPerBar ..... 4400
.....Save ..... 4401
.....Load ..... 4402
.....Type ..... 4403
.....CChartObjectGannFan..... 4404
.....Create ..... 4406
.....PipsPerBar ..... 4407
.....Downtrend ..... 4408
.....Save ..... 4409
.....Load ..... 4410
.....Type ..... 4411
.....CChartObjectGannGrid..... 4412
.....Create ..... 4414
.....PipsPerBar ..... 4415
.....Downtrend ..... 4416
.....Save ..... 4417
.....Load ..... 4418
.....Type ..... 4419
.....Fibonacci Tools..... 4420
.....CChartObjectFibo..... 4421
.....Create ..... 4422
.....Type ..... 4423
.....CChartObjectFiboTimes..... 4424
.....Create ..... 4425
.....Type ..... 4426
.....CChartObjectFiboFan..... 4427
.....Create ..... 4428
.....Type ..... 4429


© 2000-2025, MetaQuotes Ltd.

CChartObjectFiboArc 4430
Create 4432
Scale 4433
Ellipse 4434
Save 4435
Load 4436
Type 4437
CChartObjectFiboChannel 4438
Create 4439
Type 4440
CChartObjectFiboExpansion 4441
Create 4442
Type 4443
Elliott Tools 4444
CChartObjectElliottWave3 4445
Create 4447
Degree 4448
Lines 4449
Save 4450
Load 4451
Type 4452
CChartObjectElliottWave5 4453
Create 4454
Type 4456
Objects Shapes 4457
CChartObjectRectangle 4458
Create 4459
Type 4460
CChartObjectTriangle 4461
Create 4462
Type 4463
CChartObjectEllipse 4464
Create 4465
Type 4466
Objects Arrows 4467
CChartObjectArrow 4468
Create 4470
ArrowCode 4472
Anchor 4474
Save 4476
Load 4477
Type 4478
Arrows with fixed code 4479
Create 4481
ArrowCode 4483
Type 4484
Objects Controls 4485
CChartObjectText 4486
Create 4488
Angle 4489
Font 4490
FontSize 4491
Anchor 4492
Save 4493
Load 4494
Type 4495
CChartObjectLabel 4496
Create 4498

© 2000-2025, MetaQuotes Ltd.

X_Distance 4499
Y_Distance 4500
X_Size 4501
Y_Size 4502
Corner 4503
Time 4504
Price 4505
Save 4506
Load 4507
Type 4508
CChartObjectEdit 4509
Create 4511
TextAlign 4512
X_Size 4513
Y_Size 4514
BackColor 4515
BorderColor 4516
ReadOnly 4517
Angle 4518
Save 4519
Load 4520
Type 4521
CChartObjectButton 4522
State 4524
Save 4525
Load 4526
Type 4527
CChartObjectSubChart 4528
Create 4530
X_Distance 4531
Y_Distance 4532
Corner 4533
X_Size 4534
Y_Size 4535
Symbol 4536
Period 4537
Scale 4538
DateScale 4539
PriceScale 4540
Time 4541
Price 4542
Save 4543
Load 4544
Type 4545
CChartObjectBitmap 4546
Create 4548
BmpFile 4549
X_Offset 4550
Y_Offset 4551
Save 4552
Load 4553
Type 4554
CChartObjectBmpLabel 4555
Create 4557
X_Distance 4558
Y_Distance 4559
X_Offset 4560
Y_Offset 4561

© 2000-2025, MetaQuotes Ltd.

.....Corner ..... 4562
.....X_Size ..... 4563
.....Y_Size ..... 4564
.....BmpFileOn ..... 4565
.....BmpFileOff ..... 4566
.....State ..... 4567
.....Time ..... 4568
.....Price ..... 4569
.....Save ..... 4570
.....Load ..... 4571
.....Type ..... 4572
.....CChartObjectRectLabel ..... 4573
.....Create ..... 4575
.....X_Size ..... 4576
.....Y_Size ..... 4577
.....BackColor ..... 4578
.....Angle ..... 4579
.....BorderType ..... 4580
.....Save ..... 4581
.....Load ..... 4582
.....Type ..... 4583
.....自定义图形 ..... 4584
.....CCanvas ..... 4585
.....Attach ..... 4589
.....Arc ..... 4590
.....Pie ..... 4593
.....FillPolygon ..... 4596
.....FillEllipse ..... 4597
.....GetDefaultColor ..... 4598
.....ChartObjectName ..... 4599
.....Circle ..... 4600
.....CircleAA ..... 4601
.....CircleWu ..... 4602
.....Create ..... 4603
.....CreateBitmap ..... 4604
.....CreateBitmapLabel ..... 4606
.....Destroy ..... 4608
.....Ellipse ..... 4609
.....EllipseAA ..... 4610
.....EllipseWu ..... 4611
.....Erase ..... 4612
.....Fill ..... 4613
.....FillCircle ..... 4614
.....FillRectangle ..... 4615
.....FillTriangle ..... 4616
.....FontAngleGet ..... 4617
.....FontAngleSet ..... 4618
.....FontFlagsGet ..... 4619
.....FontFlagsSet ..... 4620
.....FontGet ..... 4621
.....FontNameGet ..... 4622
.....FontNameSet ..... 4623
.....FontSet ..... 4624
.....FontSizeGet ..... 4625
.....FontSizeSet ..... 4626
.....Height ..... 4627
.....Line ..... 4628
.....LineAA ..... 4629


© 2000-2025, MetaQuotes Ltd.

.....LineWu..... 4630
.....LineHorizontal..... 4631
.....LineVertical..... 4632
.....LineStyleSet..... 4633
.....LineThick..... 4634
.....LineThickVertical..... 4635
.....LineThickHorizontal..... 4636
.....LoadFromFile..... 4637
.....PixelGet..... 4638
.....PixelSet..... 4639
.....PixelSetAA..... 4640
.....Polygon..... 4641
.....PolygonAA..... 4642
.....PolygonWu..... 4643
.....PolygonThick..... 4644
.....PolygonSmooth..... 4645
.....Polyline..... 4646
.....PolylineSmooth..... 4647
.....PolylineThick..... 4648
.....PolylineWu..... 4649
.....PolylineAA..... 4650
.....Rectangle..... 4651
.....Resize..... 4652
.....ResourceName..... 4653
.....TextHeight..... 4654
.....TextOut..... 4655
.....TextSize..... 4656
.....TextWidth..... 4657
.....TransparentLevelSet..... 4658
.....Triangle..... 4659
.....TriangleAA..... 4660
.....TriangleWu..... 4661
.....Update..... 4662
.....Width..... 4663
.....CChartCanvas..... 4664
.....ColorBackground..... 4667
.....ColorBorder..... 4668
.....ColorText..... 4669
.....ColorGrid..... 4670
.....MaxData..... 4671
.....MaxDescrLen..... 4672
.....ShowFlags..... 4673
.....IsShowLegend..... 4674
.....IsShowScaleLeft..... 4675
.....IsShowScaleRight..... 4676
.....IsShowScaleTop..... 4677
.....IsShowScaleBottom..... 4678
.....IsShowGrid..... 4679
.....IsShowDescriptors..... 4680
.....IsShowPercent..... 4681
.....VScaleMin..... 4682
.....VScaleMax..... 4683
.....NumGrid..... 4684
.....DataOffset..... 4685
.....DataTotal..... 4686
.....DrawDescriptors..... 4687
.....DrawData..... 4688
.....Create..... 4689


© 2000-2025, MetaQuotes Ltd.

AllowedShowFlags 4690
ShowLegend 4691
ShowScaleLeft 4692
ShowScaleRight 4693
ShowScaleTop 4694
ShowScaleBottom 4695
ShowGrid 4696
ShowDescriptors 4697
ShowValue 4698
ShowPercent 4699
LegendAlignment 4700
Accumulative 4701
VScaleParams 4702
DescriptorUpdate 4703
ColorUpdate 4704
ValuesCheck 4705
Redraw 4706
DrawBackground 4707
DrawLegend 4708
DrawLegendVertical 4709
DrawLegendHorizontal 4710
CalcScales 4711
DrawScales 4712
DrawScaleLeft 4713
DrawScaleRight 4714
DrawScaleTop 4715
DrawScaleBottom 4716
DrawGrid 4717
DrawChart 4718
CHistogramChart 4719
Gradient 4723
BarGap 4724
BarMinSize 4725
BarBorder 4726
Create 4727
SeriesAdd 4728
SeriesInsert 4729
SeriesUpdate 4730
SeriesDelete 4731
ValueUpdate 4732
DrawData 4733
DrawBar 4734
GradientBrush 4735
CLineChart 4736
Filled 4740
Create 4741
SeriesAdd 4742
SeriesInsert 4743
SeriesUpdate 4744
SeriesDelete 4745
ValueUpdate 4746
DrawChart 4747
DrawData 4748
CalcArea 4749
CPieChart 4750
Create 4754
SeriesSet 4755
ValueAdd 4756


© 2000-2025, MetaQuotes Ltd.

ValueInsert 4757
ValueUpdate 4758
ValueDelete 4759
DrawChart 4760
DrawPie 4761
LabelMake 4762
3D图形 4763
CCanvas3D 4764
AmbientColorGet 4766
AmbientColorSet 4767
Attach 4768
Create 4769
Destroy 4770
DXContext 4771
DXDispatcher 4772
InputScene 4773
LightColorGet 4774
LightColorSet 4775
LightDirectionGet 4776
LightDirectionSet 4777
ObjectAdd 4778
ProjectionMatrixGet 4779
ProjectionMatrixSet 4780
Render 4781
RenderBegin 4782
RenderEnd 4783
ViewMatrixGet 4784
ViewMatrixSet 4785
ViewPositionSet 4786
ViewRotationSet 4787
ViewTargetSet 4788
ViewUpDirectionSet 4789
价格图表 4790
ChartID 4794
Mode 4795
Foreground 4796
Shift 4797
ShiftSize 4798
AutoScroll 4799
Scale 4800
ScaleFix 4801
ScaleFix_11 4802
FixedMax 4803
FixedMin 4804
PointsPerBar 4805
ScalePPB 4806
ShowOHLC 4807
ShowLineBid 4808
ShowLineAsk 4809
ShowLastLine 4810
ShowPeriodSep 4811
ShowGrid 4812
ShowVolumes 4813
ShowObjectDescr 4814
ShowDateScale 4815
ShowPriceScale 4816
ColorBackground 4817
ColorForeground 4818

© 2000-2025, MetaQuotes Ltd.

ColorGrid 4819
ColorBarUp 4820
ColorBarDown 4821
ColorCandleBull 4822
ColorCandleBear 4823
ColorChartLine 4824
ColorVolumes 4825
ColorLineBid 4826
ColorLineAsk 4827
ColorLineLast 4828
ColorStopLevels 4829
VisibleBars 4830
WindowsTotal 4831
WindowIsVisible 4832
WindowHandle 4833
FirstVisibleBar 4834
WidthInBars 4835
WidthInPixels 4836
HeightInPixels 4837
PriceMin 4838
PriceMax 4839
Attach 4840
FirstChart 4841
NextChart 4842
Open 4843
Detach 4844
Close 4845
BringToTop 4846
EventObjectCreate 4847
EventObjectDelete 4848
IndicatorAdd 4849
IndicatorDelete 4850
IndicatorsTotal 4851
IndicatorName 4852
Navigate 4853
Symbol 4854
Period 4855
Redraw 4856
GetInteger 4857
SetInteger 4858
GetDouble 4859
SetDouble 4860
GetString 4861
SetString 4862
SetSymbolPeriod 4863
ApplyTemplate 4864
ScreenShot 4865
WindowOnDropped 4866
PriceOnDropped 4867
TimeOnDropped 4868
XOnDropped 4869
YOnDropped 4870
Save 4871
Load 4872
Type 4873
科学图表 4874
GraphPlot 4875
CAxis 4879

© 2000-2025, MetaQuotes Ltd.

AutoScale 4880
Min 4881
Max 4882
Step 4883
Name 4884
Color 4885
ValuesSize 4886
ValuesWidth 4887
ValuesFormat 4888
NameSize 4889
ZeroLever 4890
DefaultStep 4891
MaxLabels 4892
MinGrace 4893
MaxGrace 4894
ValuesDateTimeMode 4895
ValuesFunctionFormat 4896
ValuesFunctionFormatCBData 4898
SelectAxisScale 4899
CColorGenerator 4900
Next 4901
Reset 4902
CCurve 4903
Type 4905
Name 4906
Color 4907
XMax 4908
XMin 4909
YMax 4910
YMin 4911
Size 4912
PointsSize 4913
PointsFill 4914
PointsColor 4915
GetX 4916
GetY 4917
LinesStyle 4918
LinesIsSmooth 4919
LinesSmoothTension 4920
LinesSmoothStep 4921
LinesEndStyle 4922
LinesWidth 4923
HistogramWidth 4925
CustomPlotCBData 4926
CustomPlotFunction 4927
PointsType 4931
StepsDimension 4932
TrendLineCoefficients 4933
TrendLineColor 4934
TrendLineVisible 4935
Update 4937
Visible 4939
CGraphic 4940
Create 4943
Destroy 4944
Update 4945
ChartObjectName 4946
ResourceName 4947


© 2000-2025, MetaQuotes Ltd.

XAxis 4948
YAxis 4949
GapSize 4950
BackgroundColor 4951
BackgroundMain 4952
BackgroundMainSize 4953
BackgroundMainColor 4954
BackgroundSub 4955
BackgroundSubSize 4956
BackgroundSubColor 4957
GridLineColor 4958
GridBackgroundColor 4959
GridCircleRadius 4960
GridCircleColor 4961
GridHasCircle 4962
GridAxisLineColor 4963
HistoryNameWidth 4964
HistoryNameSize 4965
HistorySymbolSize 4966
TextAdd 4967
LineAdd 4968
CurveAdd 4969
CurvePlot 4972
CurvePlotAll 4973
CurveGetByIndex 4974
CurveGetByName 4975
CurveRemoveByIndex 4976
CurveRemoveByName 4977
CurvesTotal 4978
MarksToAxisAdd 4979
MajorMarkSize 4980
FontSet 4981
FontGet 4982
Attach 4983
CalculateMaxMinValues 4984
Height 4985
IndentDown 4986
IndentLeft 4987
IndentRight 4988
IndentUp 4989
Redraw 4990
ResetParameters 4991
ScaleX 4992
ScaleY 4993
SetDefaultParameters 4994
Width 4995
指标 4996
基准类 4997
CSpreadBuffer 4998
Size 5000
SetSymbolPeriod 5001
At 5002
Refresh 5003
RefreshCurrent 5004
CTimeBuffer 5005
Size 5007
SetSymbolPeriod 5008
At 5009

© 2000-2025, MetaQuotes Ltd.

Refresh 5010
RefreshCurrent 5011
CTickVolumeBuffer 5012
Size 5014
SetSymbolPeriod 5015
At 5016
Refresh 5017
RefreshCurrent 5018
CRealVolumeBuffer 5019
Size 5021
SetSymbolPeriod 5022
At 5023
Refresh 5024
RefreshCurrent 5025
CDoubleBuffer 5026
Size 5028
SetSymbolPeriod 5029
At 5030
Refresh 5031
RefreshCurrent 5032
COpenBuffer 5033
Refresh 5034
RefreshCurrent 5035
CHighBuffer 5036
Refresh 5037
RefreshCurrent 5038
CLowBuffer 5039
Refresh 5040
RefreshCurrent 5041
CCloseBuffer 5042
Refresh 5043
RefreshCurrent 5044
CIndicatorBuffer 5045
Offset 5047
Name 5048
At 5049
Refresh 5050
RefreshCurrent 5051
CSeries 5052
Name 5054
BuffersTotal 5055
Timeframe 5056
Symbol 5057
Period 5058
RefreshCurrent 5059
BufferSize 5060
BufferResize 5061
Refresh 5062
PeriodDescription 5063
CPriceSeries 5064
BufferResize 5066
GetData 5067
Refresh 5068
MinIndex 5069
MinValue 5070
MaxIndex 5071
MaxValue 5072
CIndicator 5073


© 2000-2025, MetaQuotes Ltd.

Handle 5076
Status 5077
FullRelease 5078
Create 5079
BufferResize 5080
BarsCalculated 5081
GetData 5082
Refresh 5085
Minimum 5086
MinValue 5087
Maximum 5088
MaxValue 5089
MethodDescription 5090
PriceDescription 5091
VolumeDescription 5092
AddToChart 5093
DeleteFromChart 5094
CIndicators 5095
Create 5096
Refresh 5097
时间序列类 5098
CiSpread 5099
Create 5101
BufferResize 5102
GetData 5103
Refresh 5105
CiTime 5106
Create 5108
BufferResize 5109
GetData 5110
Refresh 5112
CiTickVolume 5113
Create 5115
BufferResize 5116
GetData 5117
Refresh 5119
CiRealVolume 5120
Create 5122
BufferResize 5123
GetData 5124
Refresh 5126
CiOpen 5127
Create 5129
GetData 5130
CiHigh 5132
Create 5134
GetData 5135
CiLow 5137
Create 5139
GetData 5140
CiClose 5142
Create 5144
GetData 5145
趋势指标 5147
CiADX 5148
MaPeriod 5150
Create 5151
Main 5152


© 2000-2025, MetaQuotes Ltd.

Plus 5153
Minus 5154
Type 5155
CiADXWilder 5156
MaPeriod 5158
Create 5159
Main 5160
Plus 5161
Minus 5162
Type 5163
CiBands 5164
MaPeriod 5166
MaShift 5167
Deviation 5168
Applied 5169
Create 5170
Base 5171
Upper 5172
Lower 5173
Type 5174
CiEnvelopes 5175
MaPeriod 5177
MaShift 5178
MaMethod 5179
Deviation 5180
Applied 5181
Create 5182
Upper 5183
Lower 5184
Type 5185
CiIchimoku 5186
TenkanSenPeriod 5188
KijunSenPeriod 5189
SenkouSpanBPeriod 5190
Create 5191
TenkanSen 5192
KijunSen 5193
SenkouSpanA 5194
SenkouSpanB 5195
ChinkouSpan 5196
Type 5197
CiMA 5198
MaPeriod 5200
MaShift 5201
MaMethod 5202
Applied 5203
Create 5204
Main 5205
Type 5206
CiSAR 5207
SarStep 5209
Maximum 5210
Create 5211
Main 5212
Type 5213
CiStdDev 5214
MaPeriod 5216
MaShift 5217


© 2000-2025, MetaQuotes Ltd.

MaMethod 5218
Applied 5219
Create 5220
Main 5221
Type 5222
CiDEMA 5223
MaPeriod 5225
IndShift 5226
Applied 5227
Create 5228
Main 5229
Type 5230
CiTEMA 5231
MaPeriod 5233
IndShift 5234
Applied 5235
Create 5236
Main 5237
Type 5238
CiFrAMA 5239
MaPeriod 5241
IndShift 5242
Applied 5243
Create 5244
Main 5245
Type 5246
CiAMA 5247
MaPeriod 5249
FastEmaPeriod 5250
SlowEmaPeriod 5251
IndShift 5252
Applied 5253
Create 5254
Main 5255
Type 5256
CiVIDyA 5257
CmoPeriod 5259
EmaPeriod 5260
IndShift 5261
Applied 5262
Create 5263
Main 5264
Type 5265
振荡器 5266
CiATR 5267
MaPeriod 5269
Create 5270
Main 5271
Type 5272
CiBearsPower 5273
MaPeriod 5275
Create 5276
Main 5277
Type 5278
CiBullsPower 5279
MaPeriod 5281
Create 5282
Main 5283

© 2000-2025, MetaQuotes Ltd.

Type 5284
CiCCI 5285
MaPeriod 5287
Applied 5288
Create 5289
Main 5290
Type 5291
CiChaikin 5292
FastMaPeriod 5294
SlowMaPeriod 5295
MaMethod 5296
Applied 5297
Create 5298
Main 5299
Type 5300
CiDeMarker 5301
MaPeriod 5303
Create 5304
Main 5305
Type 5306
CiForce 5307
MaPeriod 5309
MaMethod 5310
Applied 5311
Create 5312
Main 5313
Type 5314
CiMACD 5315
FastEmaPeriod 5317
SlowEmaPeriod 5318
SignalPeriod 5319
Applied 5320
Create 5321
Main 5322
Signal 5323
Type 5324
CiMomentum 5325
MaPeriod 5327
Applied 5328
Create 5329
Main 5330
Type 5331
CiOsMA 5332
FastEmaPeriod 5334
SlowEmaPeriod 5335
SignalPeriod 5336
Applied 5337
Create 5338
Main 5339
Type 5340
CiRSI 5341
MaPeriod 5343
Applied 5344
Create 5345
Main 5346
Type 5347
CiRVI 5348
MaPeriod 5350


© 2000-2025, MetaQuotes Ltd.

Create 5351
Main 5352
Signal 5353
Type 5354
CiStochastic 5355
Kperiod 5357
Dperiod 5358
Slowing 5359
MaMethod 5360
PriceField 5361
Create 5362
Main 5363
Signal 5364
Type 5365
CiTriX 5366
MaPeriod 5368
Applied 5369
Create 5370
Main 5371
Type 5372
CiWPR 5373
CalcPeriod 5375
Create 5376
Main 5377
Type 5378
成交量指标 5379
CiAD 5380
Applied 5382
Create 5383
Main 5384
Type 5385
CiMFI 5386
MaPeriod 5388
Applied 5389
Create 5390
Main 5391
Type 5392
CiOBV 5393
Applied 5395
Create 5396
Main 5397
Type 5398
CiVolumes 5399
Applied 5401
Create 5402
Main 5403
Type 5404
比尔·威廉姆斯指标 5405
CiAC 5406
Create 5408
Main 5409
Type 5410
CiAlligator 5411
JawPeriod 5413
JawShift 5414
TeethPeriod 5415
TeethShift 5416
LipsPeriod 5417


© 2000-2025, MetaQuotes Ltd.

.....LipsShift ..... 5418
.....MaMethod ..... 5419
.....Applied ..... 5420
.....Create ..... 5421
.....Jaw ..... 5422
.....Teeth ..... 5423
.....Lips ..... 5424
.....Type ..... 5425
...CiAO..... 5426
.....Create ..... 5428
.....Main ..... 5429
.....Type ..... 5430
...CiFractals..... 5431
.....Create ..... 5433
.....Upper ..... 5434
.....Lower ..... 5435
.....Type ..... 5436
...CiGator..... 5437
.....JawPeriod ..... 5439
.....JawShift ..... 5440
.....TeethPeriod ..... 5441
.....TeethShift ..... 5442
.....LipsPeriod ..... 5443
.....LipsShift ..... 5444
.....MaMethod ..... 5445
.....Applied ..... 5446
.....Create ..... 5447
.....Upper ..... 5448
.....Lower ..... 5449
.....Type ..... 5450
...CiBWMFI..... 5451
.....Applied ..... 5453
.....Create ..... 5454
.....Main ..... 5455
.....Type ..... 5456
...自定义指标..... 5457
...NumBuffers..... 5458
...NumParams..... 5459
...ParamType..... 5460
...ParamLong..... 5461
...ParamDouble..... 5462
...ParamString..... 5463
...Type..... 5464
交易类 .....5465
...CAccountInfo..... 5466
...Login..... 5468
...TradeMode..... 5469
...TradeModeDescription..... 5470
...Leverage..... 5471
...StopoutMode..... 5472
...StopoutModeDescription..... 5473
...MarginMode..... 5474
...MarginModeDescription..... 5475
...TradeAllowed..... 5476
...TradeExpert..... 5477
...LimitOrders..... 5478
...Balance..... 5479
...Credit..... 5480


© 2000-2025, MetaQuotes Ltd.

.....Profit..... 5481
.....Equity..... 5482
.....Margin..... 5483
.....FreeMargin..... 5484
.....MarginLevel..... 5485
.....MarginCall..... 5486
.....MarginStopOut..... 5487
.....Name..... 5488
.....Server..... 5489
.....Currency..... 5490
.....Company..... 5491
.....InfoInteger..... 5492
.....InfoDouble..... 5493
.....InfoString..... 5494
.....OrderProfitCheck..... 5495
.....MarginCheck..... 5496
.....FreeMarginCheck..... 5497
.....MaxLotCheck..... 5498
.....CSymbolInfo..... 5499
.....Refresh..... 5503
.....RefreshRates..... 5504
.....Name..... 5505
.....Select..... 5506
.....IsSynchronized..... 5507
.....Volume..... 5508
.....VolumeHigh..... 5509
.....VolumeLow..... 5510
.....Time..... 5511
.....Spread..... 5512
.....SpreadFloat..... 5513
.....TicksBookDepth..... 5514
.....StopsLevel..... 5515
.....FreezeLevel..... 5516
.....Bid..... 5517
.....BidHigh..... 5518
.....BidLow..... 5519
.....Ask..... 5520
.....AskHigh..... 5521
.....AskLow..... 5522
.....Last..... 5523
.....LastHigh..... 5524
.....LastLow..... 5525
.....TradeCalcMode..... 5526
.....TradeCalcModeDescription..... 5527
.....TradeMode..... 5528
.....TradeModeDescription..... 5529
.....TradeExecution..... 5530
.....TradeExecutionDescription..... 5531
.....SwapMode..... 5532
.....SwapModeDescription..... 5533
.....SwapRollover3days..... 5534
.....SwapRollover3daysDescription..... 5535
.....MarginInitial..... 5536
.....MarginMaintenance..... 5537
.....MarginLong..... 5538
.....MarginShort..... 5539
.....MarginLimit..... 5540
.....MarginStop..... 5541


© 2000-2025, MetaQuotes Ltd.

.....MarginStopLimit..... 5542
.....TradeTimeFlags..... 5543
.....TradeFillFlags..... 5544
.....Digits..... 5545
.....Point..... 5546
.....TickValue..... 5547
.....TickValueProfit..... 5548
.....TickValueLoss..... 5549
.....TickSize..... 5550
.....ContractSize..... 5551
.....LotsMin..... 5552
.....LotsMax..... 5553
.....LotsStep..... 5554
.....LotsLimit..... 5555
.....SwapLong..... 5556
.....SwapShort..... 5557
.....CurrencyBase..... 5558
.....CurrencyProfit..... 5559
.....CurrencyMargin..... 5560
.....Bank..... 5561
.....Description..... 5562
.....Path..... 5563
.....SessionDeals..... 5564
.....SessionBuyOrders..... 5565
.....SessionSellOrders..... 5566
.....SessionTurnover..... 5567
.....SessionInterest..... 5568
.....SessionBuyOrdersVolume..... 5569
.....SessionSellOrdersVolume..... 5570
.....SessionOpen..... 5571
.....SessionClose..... 5572
.....SessionAW..... 5573
.....SessionPriceSettlement..... 5574
.....SessionPriceLimitMin..... 5575
.....SessionPriceLimitMax..... 5576
.....InfoInteger..... 5577
.....InfoDouble..... 5578
.....InfoString..... 5579
.....NormalizePrice..... 5580
.....COrderInfo..... 5581
.....Ticket..... 5583
.....TimeSetup..... 5584
.....TimeSetupMsc..... 5585
.....OrderType..... 5586
.....TypeDescription..... 5587
.....State..... 5588
.....StateDescription..... 5589
.....TimeExpiration..... 5590
.....TimeDone..... 5591
.....TimeDoneMsc..... 5592
.....TypeFilling..... 5593
.....TypeFillingDescription..... 5594
.....TypeTime..... 5595
.....TypeTimeDescription..... 5596
.....Magic..... 5597
.....PositionId..... 5598
.....VolumeInitial..... 5599
.....VolumeCurrent..... 5600


© 2000-2025, MetaQuotes Ltd.

.....PriceOpen..... 5601
.....StopLoss..... 5602
.....TakeProfit..... 5603
.....PriceCurrent..... 5604
.....PriceStopLimit..... 5605
.....Symbol..... 5606
.....Comment..... 5607
.....InfoInteger..... 5608
.....InfoDouble..... 5609
.....InfoString..... 5610
.....StoreState..... 5611
.....CheckState..... 5612
.....Select..... 5613
.....SelectByIndex..... 5614
.....CHistoryOrderInfo..... 5615
.....TimeSetup..... 5617
.....TimeSetupMsc..... 5618
.....OrderType..... 5619
.....TypeDescription..... 5620
.....State..... 5621
.....StateDescription..... 5622
.....TimeExpiration..... 5623
.....TimeDone..... 5624
.....TimeDoneMsc..... 5625
.....TypeFilling..... 5626
.....TypeFillingDescription..... 5627
.....TypeTime..... 5628
.....TypeTimeDescription..... 5629
.....Magic..... 5630
.....PositionId..... 5631
.....VolumeInitial..... 5632
.....VolumeCurrent..... 5633
.....PriceOpen..... 5634
.....StopLoss..... 5635
.....TakeProfit..... 5636
.....PriceCurrent..... 5637
.....PriceStopLimit..... 5638
.....Symbol..... 5639
.....Comment..... 5640
.....InfoInteger..... 5641
.....InfoDouble..... 5642
.....InfoString..... 5643
.....Ticket..... 5644
.....SelectByIndex..... 5645
.....CPositionInfo..... 5646
.....Time..... 5648
.....TimeMsc..... 5649
.....TimeUpdate..... 5650
.....TimeUpdateMsc..... 5651
.....PositionType..... 5652
.....TypeDescription..... 5653
.....Magic..... 5654
.....Identifier..... 5655
.....Volume..... 5656
.....PriceOpen..... 5657
.....StopLoss..... 5658
.....TakeProfit..... 5659
.....PriceCurrent..... 5660


© 2000-2025, MetaQuotes Ltd.

.....Commission..... 5661
.....Swap..... 5662
.....Profit..... 5663
.....Symbol..... 5664
.....Comment..... 5665
.....InfoInteger..... 5666
.....InfoDouble..... 5667
.....InfoString..... 5668
.....Select..... 5669
.....SelectByIndex..... 5670
.....SelectByMagic..... 5671
.....SelectByTicket..... 5672
.....StoreState..... 5673
.....CheckState..... 5674
.....CDealInfo..... 5675
.....Order..... 5677
.....Time..... 5678
.....TimeMsc..... 5679
.....DealType..... 5680
.....TypeDescription..... 5681
.....Entry..... 5682
.....EntryDescription..... 5683
.....Magic..... 5684
.....PositionId..... 5685
.....Volume..... 5686
.....Price..... 5687
.....Commision..... 5688
.....Swap..... 5689
.....Profit..... 5690
.....Symbol..... 5691
.....Comment..... 5692
.....InfoInteger..... 5693
.....InfoDouble..... 5694
.....InfoString..... 5695
.....Ticket..... 5696
.....SelectByIndex..... 5697
.....CTrade..... 5698
.....LogLevel..... 5702
.....SetExpertMagicNumber..... 5703
.....SetDeviationInPoints..... 5704
.....SetTypeFilling..... 5705
.....SetTypeFillingBySymbol..... 5706
.....SetAsyncMode..... 5707
.....SetMarginMode..... 5708
.....OrderOpen..... 5709
.....OrderModify..... 5711
.....OrderDelete..... 5712
.....PositionOpen..... 5713
.....PositionModify..... 5714
.....PositionClose..... 5715
.....PositionClosePartial..... 5716
.....PositionCloseBy..... 5717
.....Buy..... 5718
.....Sell..... 5719
.....BuyLimit..... 5720
.....BuyStop..... 5721
.....SellLimit..... 5722
.....SellStop..... 5723


© 2000-2025, MetaQuotes Ltd.

.....Request..... 5724
.....RequestAction..... 5725
.....RequestActionDescription..... 5726
.....RequestMagic..... 5727
.....RequestOrder..... 5728
.....RequestSymbol..... 5729
.....RequestVolume..... 5730
.....RequestPrice..... 5731
.....RequestStopLimit..... 5732
.....RequestSL..... 5733
.....RequestTP..... 5734
.....RequestDeviation..... 5735
.....RequestType..... 5736
.....RequestTypeDescription..... 5737
.....RequestTypeFilling..... 5738
.....RequestTypeFillingDescription..... 5739
.....RequestTypeTime..... 5740
.....RequestTypeTimeDescription..... 5741
.....RequestExpiration..... 5742
.....RequestComment..... 5743
.....RequestPosition..... 5744
.....RequestPositionBy..... 5745
.....Result..... 5746
.....ResultRetcode..... 5747
.....ResultRetcodeDescription..... 5748
.....ResultDeal..... 5749
.....ResultOrder..... 5750
.....ResultVolume..... 5751
.....ResultPrice..... 5752
.....ResultBid..... 5753
.....ResultAsk..... 5754
.....ResultComment..... 5755
.....CheckResult..... 5756
.....CheckResultRetcode..... 5757
.....CheckResultRetcodeDescription..... 5758
.....CheckResultBalance..... 5759
.....CheckResultEquity..... 5760
.....CheckResultProfit..... 5761
.....CheckResultMargin..... 5762
.....CheckResultMarginFree..... 5763
.....CheckResultMarginLevel..... 5764
.....CheckResultComment..... 5765
.....PrintRequest..... 5766
.....PrintResult..... 5767
.....FormatRequest..... 5768
.....FormatRequestResult..... 5769
.....CTerminalInfo..... 5770
.....Build..... 5772
.....IsConnected..... 5773
.....IsDLLsAllowed..... 5774
.....IsTradeAllowed..... 5775
.....IsEmailEnabled..... 5776
.....IsFtpEnabled..... 5777
.....MaxBars..... 5778
.....CodePage..... 5779
.....CPUCores..... 5780
.....MemoryPhysical..... 5781
.....MemoryTotal..... 5782


© 2000-2025, MetaQuotes Ltd.

MemoryAvailable 5783
MemoryUsed 5784
IsX64 5785
OpenCLSupport 5786
DiskSpace 5787
Language 5788
Name 5789
Company 5790
Path 5791
DataPath 5792
CommonDataPath 5793
InfoInteger 5794
InfoString 5795
策略模块 5796
智能交易基准类 5798
CExpertBase 5799
InitPhase 5801
TrendType 5802
UsedSeries 5803
EveryTick 5804
Open 5805
High 5806
Low 5807
Close 5808
Spread 5809
Time 5810
TickVolume 5811
RealVolume 5812
Init 5813
Symbol 5814
Period 5815
Magic 5816
ValidationSettings 5817
SetPriceSeries 5818
SetOtherSeries 5819
InitIndicators 5820
InitOpen 5821
InitHigh 5822
InitLow 5823
InitClose 5824
InitSpread 5825
InitTime 5826
InitTickVolume 5827
InitRealVolume 5828
PriceLevelUnit 5829
StartIndex 5830
CompareMagic 5831
CExpert 5832
Init 5836
Magic 5837
InitSignal 5838
InitTrailing 5839
InitMoney 5840
InitTrade 5841
Deinit 5842
OnTickProcess 5843
OnTradeProcess 5844
OnTimerProcess 5845

© 2000-2025, MetaQuotes Ltd.

OnChartEventProcess 5846
OnBookEventProcess 5847
MaxOrders 5848
Signal 5849
ValidationSettings 5850
InitIndicators 5851
OnTick 5852
OnTrade 5853
OnTimer 5854
OnChartEvent 5855
OnBookEvent 5856
InitParameters 5857
DeinitTrade 5858
DeinitSignal 5859
DeinitTrailing 5860
DeinitMoney 5861
DeinitIndicators 5862
Refresh 5863
Processing 5864
SelectPosition 5866
CheckOpen 5867
CheckOpenLong 5868
CheckOpenShort 5869
OpenLong 5870
OpenShort 5871
CheckReverse 5872
CheckReverseLong 5873
CheckReverseShort 5874
ReverseLong 5875
ReverseShort 5876
CheckClose 5877
CheckCloseLong 5878
CheckCloseShort 5879
CloseAll 5880
Close 5881
CloseLong 5882
CloseShort 5883
CheckTrailingStop 5884
CheckTrailingStopLong 5885
CheckTrailingStopShort 5886
TrailingStopLong 5887
TrailingStopShort 5888
CheckTrailingOrderLong 5889
CheckTrailingOrderShort 5890
TrailingOrderLong 5891
TrailingOrderShort 5892
CheckDeleteOrderLong 5893
CheckDeleteOrderShort 5894
DeleteOrders 5895
DeleteOrder 5896
DeleteOrderLong 5897
DeleteOrderShort 5898
LotOpenLong 5899
LotOpenShort 5900
LotReverse 5901
PrepareHistoryDate 5902
HistoryPoint 5903
CheckTradeState 5904

© 2000-2025, MetaQuotes Ltd.

WaitEvent 5905
NoWaitEvent 5906
TradeEventPositionStopTake 5907
TradeEventOrderTriggered 5908
TradeEventPositionOpened 5909
TradeEventPositionVolumeChanged 5910
TradeEventPositionModified 5911
TradeEventPositionClosed 5912
TradeEventOrderPlaced 5913
TradeEventOrderModified 5914
TradeEventOrderDeleted 5915
TradeEventNotIdentified 5916
TimeframeAdd 5917
TimeframesFlags 5918
CExpertSignal 5919
BasePrice 5922
UsedSeries 5923
Weight 5924
PatternsUsage 5925
General 5926
Ignore 5927
Invert 5928
ThresholdOpen 5929
ThresholdClose 5930
PriceLevel 5931
StopLevel 5932
TakeLevel 5933
Expiration 5934
Magic 5935
ValidationSettings 5936
InitIndicators 5937
AddFilter 5938
CheckOpenLong 5939
CheckOpenShort 5940
OpenLongParams 5941
OpenShortParams 5942
CheckCloseLong 5943
CheckCloseShort 5944
CloseLongParams 5945
CloseShortParams 5946
CheckReverseLong 5947
CheckReverseShort 5948
CheckTrailingOrderLong 5949
CheckTrailingOrderShort 5950
LongCondition 5951
ShortCondition 5952
Direction 5953
CExpertTrailing 5954
CheckTrailingStopLong 5955
CheckTrailingStopShort 5956
CExpertMoney 5957
Percent 5958
ValidationSettings 5959
CheckOpenLong 5960
CheckOpenShort 5961
CheckReverse 5962
CheckClose 5963
交易信号模块 5964


© 2000-2025, MetaQuotes Ltd.

加速计振荡器指标信号 5967
自适应移动均线指标信号 5969
动量振荡器指标信号 5973
空头力度振荡器信号 5976
多头力度振荡器信号 5978
商品通道指数振荡器信号 5980
DeMarker 振荡器信号 5984
双重指数移动均线指标信号 5988
包络线指标信号 5992
分形自适应移动均线指标信号 5995
日内时间过滤信号 5999
MACD 振荡器信号 6000
移动均线指标信号 6006
抛物线转向指标信号 6010
相对强度指数振荡器信号 6012
相对动量指数振荡器信号 6018
随机振荡器信号 6020
三重指数均线振荡器信号 6025
三重指数移动均线指标信号 6029
威廉姆斯百分比范围振荡器信号 6033
尾随停止类 6036
CTrailingFixedPips 6037
StopLevel 6039
ProfitLevel 6040
ValidationSettings 6041
CheckTrailingStopLong 6042
CheckTrailingStopShort 6043
CTrailingMA 6044
Period 6045
Shift 6046
Method 6047
Applied 6048
InitIndicators 6049
ValidationSettings 6050
CheckTrailingStopLong 6051
CheckTrailingStopShort 6052
CTrailingNone 6053
CheckTrailingStopLong 6054
CheckTrailingStopShort 6055
CTrailingPSAR 6056
Step 6057
Maximum 6058
InitIndicators 6059
CheckTrailingStopLong 6060
CheckTrailingStopShort 6061
资金管理类 6062
CMoneyFixedLot 6063
Lots 6064
ValidationSettings 6065
CheckOpenLong 6066
CheckOpenShort 6067
CMoneyFixedMargin 6068
CheckOpenLong 6069
CheckOpenShort 6070
CMoneyFixedRisk 6071
CheckOpenLong 6072
CheckOpenShort 6073
CMoneyNone 6074


© 2000-2025, MetaQuotes Ltd.

ValidationSettings 6075
CheckOpenLong 6076
CheckOpenShort 6077
CMoneySizeOptimized 6078
DecreaseFactor 6079
ValidationSettings 6080
CheckOpenLong 6081
CheckOpenShort 6082
面板和对话框 6083
CRect 6085
Left 6086
Top 6087
Right 6088
Bottom 6089
Width 6090
Height 6091
SetBound 6092
Move 6093
Shift 6094
Contains 6095
Format 6096
CDateTime 6097
MonthName 6099
ShortMonthName 6100
DayName 6101
ShortDayName 6102
DaysInMonth 6103
DateTime 6104
Date 6105
Time 6106
Sec 6107
Min 6108
Hour 6109
Day 6110
Mon 6111
Year 6112
SecDec 6113
SecInc 6114
MinDec 6115
MinInc 6116
HourDec 6117
HourInc 6118
DayDec 6119
DayInc 6120
MonDec 6121
MonInc 6122
YearDec 6123
YearInc 6124
CWnd 6125
Create 6129
Destroy 6130
OnEvent 6131
OnMouseEvent 6132
Name 6133
ControlsTotal 6134
Control 6135
ControlFind 6136
Rect 6137


© 2000-2025, MetaQuotes Ltd.

.....Left..... 6138
.....Top..... 6139
.....Right..... 6140
.....Bottom..... 6141
.....Width..... 6142
.....Height..... 6143
.....Move..... 6144
.....Shift..... 6145
.....Resize..... 6146
.....Contains..... 6147
.....Alignment..... 6148
.....Align..... 6149
.....Id..... 6150
.....IsEnabled..... 6151
.....Enable..... 6152
.....Disable..... 6153
.....IsVisible..... 6154
.....Visible..... 6155
.....Show..... 6156
.....Hide..... 6157
.....IsActive..... 6158
.....Activate..... 6159
.....Deactivate..... 6160
.....StateFlags..... 6161
.....StateFlagsSet..... 6162
.....StateFlagsReset..... 6163
.....PropFlags..... 6164
.....PropFlagsSet..... 6165
.....PropFlagsReset..... 6166
.....MouseX..... 6167
.....MouseY..... 6168
.....MouseFlags..... 6169
.....MouseFocusKill..... 6170
.....OnCreate..... 6171
.....OnDestroy..... 6172
.....OnMove..... 6173
.....OnResize..... 6174
.....OnEnable..... 6175
.....OnDisable..... 6176
.....OnShow..... 6177
.....OnHide..... 6178
.....OnActivate..... 6179
.....OnDeactivate..... 6180
.....OnClick..... 6181
.....OnChange..... 6182
.....OnMouseDown..... 6183
.....OnMouseUp..... 6184
.....OnDragStart..... 6185
.....OnDragProcess..... 6186
.....OnDragEnd..... 6187
.....DragObjectCreate..... 6188
.....DragObjectDestroy..... 6189
.....CWndObj..... 6190
.....OnEvent..... 6192
.....Text..... 6193
.....Color..... 6194
.....ColorBackground..... 6195
.....ColorBorder..... 6196


© 2000-2025, MetaQuotes Ltd.

.....Font..... 6197
.....FontSize..... 6198
.....ZOrder..... 6199
.....OnObjectCreate..... 6200
.....OnObjectChange..... 6201
.....OnObjectDelete..... 6202
.....OnObjectDrag..... 6203
.....OnSetText..... 6204
.....OnSetColor..... 6205
.....OnSetColorBackground..... 6206
.....OnSetFont..... 6207
.....OnSetFontSize..... 6208
.....OnSetZOrder..... 6209
.....OnDestroy..... 6210
.....OnChange..... 6211
.....CWndContainer..... 6212
.....Destroy..... 6214
.....OnEvent..... 6215
.....OnMouseEvent..... 6216
.....ControlsTotal..... 6217
.....Control..... 6218
.....ControlFind..... 6219
.....Add..... 6220
.....Delete..... 6221
.....Move..... 6222
.....Shift..... 6223
.....Id..... 6224
.....Enable..... 6225
.....Disable..... 6226
.....Show..... 6227
.....Hide..... 6228
.....MouseFocusKill..... 6229
.....Save..... 6230
.....Load..... 6231
.....OnResize..... 6232
.....OnActivate..... 6233
.....OnDeactivate..... 6234
.....CLabel..... 6235
.....Create..... 6240
.....OnSetText..... 6241
.....OnSetColor..... 6242
.....OnSetFont..... 6243
.....OnSetFontSize..... 6244
.....OnCreate..... 6245
.....OnShow..... 6246
.....OnHide..... 6247
.....OnMove..... 6248
.....CBmpButton..... 6249
.....Create..... 6255
.....Border..... 6256
.....BmpNames..... 6257
.....BmpOffName..... 6258
.....BmpOnName..... 6259
.....BmpPassiveName..... 6260
.....BmpActiveName..... 6261
.....Pressed..... 6262
.....Locking..... 6263
.....OnSetZOrder..... 6264


© 2000-2025, MetaQuotes Ltd.

OnCreate 6265
OnShow 6266
OnHide 6267
OnMove 6268
OnChange 6269
OnActivate 6270
OnDeactivate 6271
OnMouseDown 6272
OnMouseUp 6273
CButton 6274
Create 6281
Pressed 6282
Locking 6283
OnSetText 6284
OnSetColor 6285
OnSetColorBackground 6286
OnSetColorBorder 6287
OnSetFont 6288
OnSetFontSize 6289
OnCreate 6290
OnShow 6291
OnHide 6292
OnMove 6293
OnResize 6294
OnMouseDown 6295
OnMouseUp 6296
CEdit 6297
Create 6302
ReadOnly 6303
TextAlign 6304
OnObjectEndEdit 6305
OnSetText 6306
OnSetColor 6307
OnSetColorBackground 6308
OnSetColorBorder 6309
OnSetFont 6310
OnSetFontSize 6311
OnSetZOrder 6312
OnCreate 6313
OnShow 6314
OnHide 6315
OnMove 6316
OnResize 6317
OnChange 6318
OnClick 6319
CPanel 6320
Create 6325
BorderType 6326
OnSetText 6327
OnSetColorBackground 6328
OnSetColorBorder 6329
OnCreate 6330
OnShow 6331
OnHide 6332
OnMove 6333
OnResize 6334
OnChange 6335
CPicture 6336


© 2000-2025, MetaQuotes Ltd.

.....Create..... 6341
.....Border..... 6342
.....BmpName..... 6343
.....OnCreate..... 6344
.....OnShow..... 6345
.....OnHide..... 6346
.....OnMove..... 6347
.....OnChange..... 6348
.....CScroll..... 6349
.....Create..... 6351
.....OnEvent..... 6352
.....MinPos..... 6353
.....MaxPos..... 6354
.....CurrPos..... 6355
.....CreateBack..... 6356
.....CreateInc..... 6357
.....CreateDec..... 6358
.....CreateThumb..... 6359
.....OnClickInc..... 6360
.....OnClickDec..... 6361
.....OnShow..... 6362
.....OnHide..... 6363
.....OnChangePos..... 6364
.....OnThumbDragStart..... 6365
.....OnThumbDragProcess..... 6366
.....OnThumbDragEnd..... 6367
.....CalcPos..... 6368
.....CScrollV..... 6369
.....CreateInc..... 6375
.....CreateDec..... 6376
.....CreateThumb..... 6377
.....OnResize..... 6378
.....OnChangePos..... 6379
.....OnThumbDragStart..... 6380
.....OnThumbDragProcess..... 6381
.....OnThumbDragEnd..... 6382
.....CalcPos..... 6383
.....CScrollH..... 6384
.....CreateInc..... 6390
.....CreateDec..... 6391
.....CreateThumb..... 6392
.....OnResize..... 6393
.....OnChangePos..... 6394
.....OnThumbDragStart..... 6395
.....OnThumbDragProcess..... 6396
.....OnThumbDragEnd..... 6397
.....CalcPos..... 6398
.....CWndClient..... 6399
.....Create..... 6401
.....OnEvent..... 6402
.....ColorBackground..... 6403
.....ColorBorder..... 6404
.....BorderType..... 6405
.....VScrolled..... 6406
.....HScrolled..... 6407
.....CreateBack..... 6408
.....CreateScrollV..... 6409
.....CreateScrollH..... 6410


© 2000-2025, MetaQuotes Ltd.

OnResize 6411
OnVScrollShow 6412
OnVScrollHide 6413
OnHScrollShow 6414
OnHScrollHide 6415
OnScrollLineDown 6416
OnScrollLineUp 6417
OnScrollLineLeft 6418
OnScrollLineRight 6419
Rebound 6420
CListView 6421
Create 6427
OnEvent 6428
TotalView 6429
AddItem 6430
Select 6431
SelectByText 6432
SelectByValue 6433
Value 6434
CreateRow 6435
OnResize 6436
OnVScrollShow 6437
OnVScrollHide 6438
OnScrollLineDown 6439
OnScrollLineUp 6440
OnItemClick 6441
Redraw 6442
RowState 6443
CheckView 6444
CComboBox 6445
Create 6451
OnEvent 6452
AddItem 6453
ListViewItems 6454
Select 6455
SelectByText 6456
SelectByValue 6457
Value 6458
CreateEdit 6459
CreateButton 6460
CreateList 6461
OnClickEdit 6462
OnClickButton 6463
OnChangeList 6464
ListShow 6465
ListHide 6466
CCheckBox 6467
Create 6473
OnEvent 6474
Text 6475
Color 6476
Checked 6477
Value 6478
CreateButton 6479
CreateLabel 6480
OnClickButton 6481
OnClickLabel 6482
CCheckGroup 6483


© 2000-2025, MetaQuotes Ltd.

.....Create..... 6489
.....OnEvent..... 6490
.....AddItem..... 6491
.....Value..... 6492
.....CreateButton..... 6493
.....OnVScrollShow..... 6494
.....OnVScrollHide..... 6495
.....OnScrollLineDown..... 6496
.....OnScrollLineUp..... 6497
.....OnChangeItem..... 6498
.....Redraw..... 6499
.....RowState..... 6500
.....CRadioButton..... 6501
.....Create..... 6503
.....OnEvent..... 6504
.....Text..... 6505
.....Color..... 6506
.....State..... 6507
.....CreateButton..... 6508
.....CreateLabel..... 6509
.....OnClickButton..... 6510
.....OnClickLabel..... 6511
.....CRadioGroup..... 6512
.....Create..... 6518
.....OnEvent..... 6519
.....AddItem..... 6520
.....Value..... 6521
.....CreateButton..... 6522
.....OnVScrollShow..... 6523
.....OnVScrollHide..... 6524
.....OnScrollLineDown..... 6525
.....OnScrollLineUp..... 6526
.....OnChangeItem..... 6527
.....Redraw..... 6528
.....RowState..... 6529
.....Select..... 6530
.....CSpinEdit..... 6531
.....Create..... 6536
.....OnEvent..... 6537
.....MinValue..... 6538
.....MaxValue..... 6539
.....Value..... 6540
.....CreateEdit..... 6541
.....CreateInc..... 6542
.....CreateDec..... 6543
.....OnClickInc..... 6544
.....OnClickDec..... 6545
.....OnChangeValue..... 6546
.....CDialog..... 6547
.....Create..... 6549
.....OnEvent..... 6550
.....Caption..... 6551
.....Add..... 6552
.....CreateWhiteBorder..... 6553
.....CreateBackground..... 6554
.....CreateCaption..... 6555
.....CreateButtonClose..... 6556
.....CreateClientArea..... 6557


© 2000-2025, MetaQuotes Ltd.

.....OnClickCaption..... 6558
.....OnClickButtonClose..... 6559
.....ClientAreaVisible..... 6560
.....ClientAreaLeft..... 6561
.....ClientAreaTop..... 6562
.....ClientAreaRight..... 6563
.....ClientAreaBottom..... 6564
.....ClientAreaWidth..... 6565
.....ClientAreaHeight..... 6566
.....OnDialogDragStart..... 6567
.....OnDialogDragProcess..... 6568
.....OnDialogDragEnd..... 6569
.....CAppDialog..... 6570
.....Create..... 6573
.....Destroy..... 6574
.....OnEvent..... 6575
.....Run..... 6576
.....ChartEvent..... 6577
.....Minimized..... 6578
.....IniFileSave..... 6579
.....IniFileLoad..... 6580
.....IniFileName..... 6581
.....IniFileExt..... 6582
.....CreateCommon..... 6583
.....CreateExpert..... 6584
.....CreateIndicator..... 6585
.....CreateButtonMinMax..... 6586
.....OnClickButtonClose..... 6587
.....OnClickButtonMinMax..... 6588
.....OnAnotherApplicationClose..... 6589
.....Rebound..... 6590
.....Minimize..... 6591
.....Maximize..... 6592
.....CreateInstanceId..... 6593
.....ProgramName..... 6594
.....SubwinOff..... 6595
.....36．来自 MQL4..... 6596
.....37．MQL5 函数列表..... 6599
.....38．MQL5 常量列表..... 6623


© 2000-2025, MetaQuotes Ltd.

# MQL5参考

MetaQuotes Language 5 (MQL5)是一种高水平编程语言，专为开发技术指标、EA交易和实用应用程序而设计，进而使金融交易自动化。MQL5是由MetaQuotes为其交易平台而开发的。这个语言的语法非常接近于C++，使程序员能够在面向对象编程(OOP)风格中开发应用程序。

除了MQL5语言之外，交易平台程序包还包括高级代码编写工具MetaEditor IDE，如模板、代码片段、调试、概要分析和自动完成工具，以及内置的支持文件版本的MQL5存储。

语言支持还可用在MQL5 Algotrading community网站，其包含一个大型免费代码库和大量的文章。这些文章涵盖了现代交易的所有方面，包括神经网络、统计和分析、高频交易、套利、测试和优化交易策略、使用EA交易等。

交易员和MQL5程序开发人员可以在论坛上进行沟通，使用自由职业者服务下订单并开发应用程序，以及在自动交易应用程序市场中购买和出售受保护的程序。

MQL5语言提供了专门的交易函数和预定义事件处理程序，以帮助程序员开发EA交易，根据特定的交易规则自动控制交易流程。除了EA交易之外，MQL5还可以开发自定义技术指标、脚本和程序库。

MQL5语言参考包括函数，操作，关键字和分为几类的其他语言架构。该参考还提供了用于开发交易策略、控制面板、自定义图形和启用文件访问的标准程序库类的描述。

另外，代码库还包含了ALGLIB数值分析库，可被用于解决各种数学问题。

## 算法交易书籍

开始学习新东西总是具有挑战性的。为了帮助初学者，我们发布了两本关于 MQL5 编程的综合书籍，专为希望掌握创建交易机器人和算法交易应用程序的人而设计。

这些书对材料进行了系统和结构化的介绍，使学习过程变得更加容易。详细的代码示例解释了交易机器人和应用程序的逐步创建，使读者可以更深入地理解算法交易的细节。这些书包括许多实践练习，以帮助加强所获得的知识，并在真实的交易环境中培养编程技能。

"面向交易者的MQL5编程"是关于MQL5的最完整、最详细的教程，适合各级程序开发人员。初学者将学习基础知识：本书介绍了开发工具和基本编程概念。基于这些材料，您将在 MetaTrader 5 交易平台中创建、编译和运行您的第一个应用程序。具有其他编程语言经验的用户可以立即进入应用程序部分：在 MQL5 中创建交易机器人和分析应用程序。

"MQL5 算法交易的神经网络"是 MetaTrader 5 平台交易机器人中使用机器学习方法的指南。您将逐步了解神经网络的基本原理及其在算法交易中的应用。随着您的进步，您将构建和训练自己的人工智能解决方案，逐步添加新功能。除了学习 MQL5，您还将获得 Python 和 OpenCL 编程技能，并探索集成矩阵和向量方法，从而能够用简洁高效的代码解决复杂的数学问题。

## 关于交易应用程序开发的文章

MQL5 文章是探索该语言全部潜力的绝佳资源，涵盖了广泛的实用算法交易任务。为了便于导航，所有文章都分为以下类别：示例、EA 交易、机器学习等等。每个月， 都会有数十篇由交易员为其他交易员撰写的新文章发布在MQL5 算法交易社区网站上。阅读并讨论这些文章，掌握现

© 2000-2025, MetaQuotes Ltd.

代算法交易。对于初学者，我们编制了一份16篇推荐文章的列表，以便快速掌握MQL5。

## MQL5应用程序类型

MQL5程序根据实施的交易自动化任务，被分成四种特定类型：

- **EA交易** 是一种与图表相链接的自动交易系统。EA交易包含了管理预定义事件的事件处理程序，激活执行适当的交易策略元素。例如，程序初始化和去初始化的事件，新报价，计时器事件、市场深度变化、图表和自定义事件。
  除了根据已实现的规则计算基于信号之外，EA交易还能够自动执行交易，并将其直接发送到交易服务器。EA交易被存储在*<Terminal_Directory>\MQL5\Experts*。
- **自定义指标**是一种除了集成到交易平台上的标准指标之外的由用户开发的技术指标。自定义指标，以及标准指标都不能自动交易，但只能实现分析功能。自定义指标可以利用其它指标的值进行计算，也可以从EA交易调用。
  自定义指标被存储在*<Terminal_Directory>\MQL5\Indicators*。
- **脚本**是一种单独执行操作的程序。与EA交易不同，除了触发器、初始化和去初始化之外，脚本不处理任何事件。脚本代码必须包括OnStart处理程序函数。
  脚本被存储在*<Terminal_DIrectory>\MQL5\Scripts*中。
- **服务**是一个程序，与指标、EA交易和脚本不同，它不需要绑定图表进行工作。与脚本一样，除了触发器，服务不处理任何事件。要启动一项服务，其代码应该包含OnStart处理程序函数。
  除了“Start（开始）”以外，服务不接受任何其他事件，但它们可以使用EventChartCustom将自定义事件发送至图表。服务存储在*<Terminal_Directory>\MQL5\Services*。
- **程序库**是一组自定义函数。程序库用来存储和分配常用的自定义程序算法。
  程序库被存储在 *<Terminal_Directory>\MQL5\Libraries*。
- **包含文件**是最常使用的自定义程序块的源文本。这些文件存在于编译阶段的EA交易、脚本、自定义指标和程序库的源文本中。使用包含文件比使用程序库更可取，因为在调用程序库函数时容易发生额外的负担。
  包含文件可以存储在与源文件相同的目录中。在这种情况下，使用双引号的#include指令。另一个选项就是将包含文件存储在*<Terminal_Directory>\MQL5\Include*。在这种情况下，#include应该使用尖括号。

© 2000-2025, MetaQuotes Ltd

# 语言基础

MetaQuotes Language 5(MQL5)是一种面向对象的高水平的程序语言，它用来自动录入交易战略，为金融市场的各种分析定制智能指标。它不仅允许录入各种智能系统，更致力于实践操作，还能建立专属的图表工具帮您制定交易决策。

MQL5是以最流行的程序C++语言为蓝本的，相比与MQL4，新语言拥有计数，结构，分类 和 事件处理功能。通过增加嵌入式主要标签的数量，在MQL5的可执行程序的相互作用下，其他应用的运行就相对容易得多，MQL5的语法与C++的语法相似，这样一来，把现代程序语言译成它自己的语言就很容易。

为帮助您学习MQL系列语言，所有主题可分为如下步骤：

- 语法
- 数据类型
- 运行式和表达式
- 运算符
- 函数
- 变量
- 预处理程序
- 面向对象的程序设计
- 命名空间

© 2000-2025, MetaQuotes Ltd.

# 语法

至于语法，MQL5语言为程序交易策略语言与C++编程语言十分类似，除了如下特点：

- 没有运算地址；
- 没有goto语句；
- 匿名计算无法删除；
- 无成倍继承.

查看相关

计数，结构和类，继承

© 2000-2025, MetaQuotes Ltd.

# 注释

多行注释使用 /*作为开始到 */ 结束，在这之间不能够嵌套。单行注释使用 // 作为开始到新的一行结束，可以被嵌套到多行注释之中。

示例：

```
//--- 单行线注释
/*  Multi-
    line          // 嵌入式单行线注释
    comment
*/
```

© 2000-2025, MetaQuotes Ltd.

# 标识符

标识符用来给变量和函数进行命名，长度不能超过63个字节。

特属性够输入到标识符中，你可以使用数字0-9、拉丁字母大写A-Z和小写a-z(大小写有区分的)还有下划线(_)。此外首字母不可以是数字，标识符不能和 保留字冲突。

示例：

```
NAME1 name1 Total_5 Paper
```

查看相关

变量，函数

© 2000-2025, MetaQuotes Ltd.

# 关键词

下面列出的是固定的保留字标识符，每个人标识符相当于一个动作，不能用来操作其他命令。

数据类型

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

访问分类符

| | | |
|---|---|---|
| const | private | virtual |
| delete | protected | |
| override | public | |

存储类型

| | | |
|---|---|---|
| extern | input | static |

操作符

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

其他

| | | |
|---|---|---|
| this | #define | #import |
| true | #ifdef | #include |

© 2000-2025, MetaQuotes Ltd.

| | | |
|---|---|---|
| this | #define | #import |
| false | #ifndef | #property |
| template | #else | group |
| typename | #endif | namespace |

© 2000-2025, MetaQuotes Ltd.

# 数据类型

任何程序都依靠数据来运作，数据因目的不同可以有不同的类型 。比如，访问数组可以用整型数据，价格可以用双精度的浮点型数据。在 MQL5 中没有专门用来标记货币值的数据类型。

不同的数据类型有不同的处理速度，整型数据是最快的。双精度的数据处理需要特殊的协同处理器，然而，因为处理浮点型数据比较复杂，所以它比处理整型数据慢一些。

字符串是处理速度最慢的，因为它要存取动态内存。

基本数据类型：

- 整型数据 (char, short, int, long, uchar, ushort, uint, ulong)
- 逻辑数据 (bool);
- 字符数据 (ushort);
- 字符串数据 (string);
- 浮点型数据 (double, float);
- 颜色 (color);
- 日期和时间数据 (datetime);
- 计数 (enum).

复杂的数据类型是：

- structures;
- classes.

根据OOP复杂数据类型被称作抽象数据类型。

颜色 和 时间日期 从外观上可以使我们更清楚的区分图表中的内容和参数的录入，在EA交易和自定义指标中经常使用这些数据类型 (Inputs 标签)。颜色和日期时间数据用整数来表示。整型数据和浮点数据都属于数值 (数字)型。

**只有在**公式 中使用类型分类，除非提供指定强制类。

另见

类型分类

© 2000-2025, MetaQuotes Ltd.

# 整型数据

在MQL5中整数有11个类型，如果逻辑程序需要，一些类型能与另一些类型一起使用，但是在此种情况下，要记住类型转换规则。

下面列表中显示了每一类型的特性，此外，每一类型的最后一列均与C++程序类型相同。

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

整型值也可视为数字常数，颜色值和，日期-时间值，字符常量和计数。

## 其他整数类型别名 (用于 C/C++ 兼容性)

为了简化从其他语言 (如 C 和 C++)移植代码，MQL5 语言中添加了标准整数类型的别名。这些别名不会引入新类型，而是 MQL5 中现有类型的替代名称。它们可以在所有适用于基础类型的情况下使用。

| 类型 | 对应于 MQL5 类型 | 大小 (字节) | 符号 | 最小值 | 最大值 |
|---|---|---|---|---|---|
| int8_t | char | 8 | 有符号 | -128 | 127 |
| uint8_t | uchar | 8 | 无符号 | 0 | 255 |
| int16_t | short | 16 | 有符号 | -32 768 | 32 767 |

© 2000-2025, MetaQuotes Ltd.

| 类型 | 对应于MQL5类型 | 大小（字节） | 符号 | 最小值 | 最大值 |
|---|---|---|---|---|---|
| uint16_t | ushort | 16 | 无符号 | 0 | 65 535 |
| int32_t | int | 32 | 有符号 | -2 147 483 648 | 2 147 483 647 |
| uint32_t | uint | 32 | 无符号 | 0 | 4 294 967 295 |
| int64_t | long | 64 | 有符号 | -9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 |
| uint64_t | ulong | 64 | 无符号 | 0 | 18 446 744 073 709 551 615 |

这些别名在将现有库和算法从C/C++移植到MQL5时非常有用，特别是在具有高度平台兼容性的项目中。所有新名称都可以在脚本、EA和指标中使用。

另见

数据变换，数值类型常量

© 2000-2025, MetaQuotes Ltd.

# 字符型，短整型，整型和长整型

## char

*字符型*在内存里存储1字节（8位元组），允许二进制记录法2^8=256表达值。*字符型*包括正值和负值，范围在-128-127之间。

## uchar

*无字符型*和*字符型*一样，也占据1字节内存，但与之不同的是，它只有正值。最小值是0，最大值是255，*无字符型*的第一个字母u是*unsigned*的缩写。

## short

*短型*数据2字节（16位元组），所以它可以表达等于2的值和16：2^16 = 65 536，因此*短型*还是一个符号，包括正值和负值，范围在-32768到32767之间。

## ushort

*无符短型*是*ushort*，也占用2字节，最小值是0，最大值是65535。.

## int

*整型*占用4字节内存（32元组），最小值是-2 147 483 648，最大值是2 147 483 647。

## uint

*无符号整型*就是*uint*。它占用4字节内存，区间在0到4 294 967 295之间。

## long

*长型*占用8字节（64元组），最小值是-9 223 372 036 854 775 808，最大值是9 223 372 036 854 775 807。

## ulong

*无符长型*也占用8字节，能存储从0到18 446 744 073 709 551 615之间的值。

示例：

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

无符长型不代表短型的负值，建立负值会导致意外的结果，该脚本会无限循环：

```
//--- 无限循环
void OnStart()
  {

```

© 2000-2025, MetaQuotes Ltd.

```
   uchar  u_ch;

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

正确的转化是：

```
//--- 正确的变量
void OnStart()
  {
   uchar  u_ch;

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

结果：

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

示例：

```
//--- 负值不能存储在无符类型中
uchar  u_ch=-120;
ushort u_sh=-5000;
```

© 2000-2025, MetaQuotes Ltd.

```
uint   u_in=-401280;
```

十六进制：数字0-9，字母a-f，或A-F表示值10-15，从0x或0X开始。

示例：

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

对于整型变量，值可以使用B前缀以二进制格式设置。例如，您可以编码交易时间的工作小时为int类型变量并根据所需的算法使用他们的信息：

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置 1 为工作小时和 0 为非工作小时
   int AsianSession   =B'111111111'; // 亚洲时间从 0:00 到 9:00
   int EuropeanSession=B'111111111000000000'; // 欧洲时间 9:00 - 18:00
   int AmericanSession =B'11111111000000000000000011'; // 美洲时间 16:00 - 02:00
//--- 派生时间数值
   PrintFormat("Asian session hours as value =%d",AsianSession);
   PrintFormat("European session hours as value is %d",EuropeanSession);
   PrintFormat("American session hours as value is %d",AmericanSession);
//--- 现在让我们展示时间工作小时的字符串表示
   Print("Asian session ",GetHoursForSession(AsianSession));
   Print("European session ",GetHoursForSession(EuropeanSession));
   Print("American session ",GetHoursForSession(AmericanSession));
//---
  }
//+------------------------------------------------------------------+
//| 返回时间工作小时为字符串                                         |
//+------------------------------------------------------------------+
string GetHoursForSession(int session)
  {
//--- 若要检查，使用AND比特操作并左移1比特<<=1
//--- 从最小比特开始检查
   int bit=1;
   string out="working hours: ";
//--- 从零开始检查全部24比特并最高到23
   for(int i=0;i<24;i++)
     {
      //--- 总共接收的比特数
      bool workinghour=(session&bit)==bit;
      //--- 添加小时数到嘻嘻
      if(workinghour )out=out+StringFormat("%d ",i);
      //--- 左移1比特检查下一个的值
      bit<<=1;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 结果字符串
   return out;
  }
```

另见：

类型转换

© 2000-2025, MetaQuotes Ltd.

# 字符常量

字符作为MQL5中的字符串要素是统一字符设置的指数。它们是可以转换成整数的十六进制值，可以像加减法定理一样进行整数操作。

引号里的任何单一特质和十六进制的 ASCII 代码'\x10'都能当做字符常量无符号短整型 型，例如，0型记录的数值是30，相当于在图标字符中代表调整归零。

示例：

```
void OnStart()
  {
//--- 定义字符常量
   int symbol_0='0';
   int symbol_9=symbol_0+9; // 获得符号 '9'
//--- 输出常量值
   printf("In a decimal form: symbol_0 = %d,  symbol_9 = %d",symbol_0,symbol_9);
   printf("In a hexadecimal form: symbol_0 = 0x%x,  symbol_9 = 0x%x",symbol_0,symbol_9
//--- 输入常量成字符串
   string test="";
   StringSetCharacter(test,0,symbol_0);
   StringSetCharacter(test,1,symbol_9);
//--- 这是字符串中看起来像的东西
   Print(test);
  }
```

在程序源文本与常量型进行处理时，反斜线符号是编译器的控制字符，，一些符号，例如，单引号 (')，双引号 (")，反斜杠 (\)和控制字符都能被当做以反斜杠 (\)为起点的符号的集合，如下表：

| 字符名称 | 助记码或者图像 | MQL5中的记录 | 数值 |
|---|---|---|---|
| 新线（换行） | LF | '\n' | 10 |
| 水平制表符 | HT | '\t' | 9 |
| 回车 | CR | '\r' | 13 |
| 反斜杠 | \ | '\\' | 92 |
| 单引号 | ' | '\'' | 39 |
| 双引号 | " | '\"' | 34 |
| 十六进制代码 | hhhh | '\xhhhh' | 1 to 4 十六进制字符 |
| 十进制代码 | d | '\d' | 从 0 到 65535的十进制代码 |

如果反斜杠后面跟着另一种字符而不是以上描述的类型，结果是未知的。

示例

```
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 声明字符常量
   int a='A';
   int b='$';
   int c='©';       // 代码 0xA9
   int d='\xAE';    // 符号代码 ®
//--- 输出打印常量
   Print(a,b,c,d);
//--- 添加字符到字符串
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,b);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,c);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,d);
   Print(test);
//--- 字符表示为数字
   int a1=65;
   int b1=36;
   int c1=169;
   int d1=174;
//--- 添加字符到字符串
   StringSetCharacter(test,1,a1);
   Print(test);
//--- 添加字符到字符串
   StringSetCharacter(test,1,b1);
   Print(test);
//--- 添加字符到字符串
   StringSetCharacter(test,1,c1);
   Print(test);
//--- 添加字符到字符串
   StringSetCharacter(test,1,d1);
   Print(test);
  }
```

综上所述，常量（或变量）字符的值是字符表中的指标，指标存在整型，能以不同的方式输出。

```
void OnStart()
  {
//---
   int a=0xAE;      // 代码 ® 符合于 the '\xAE' 文字
   int b=0x24;      // 代码 $ 符合于 the '\x24' 文字
   int c=0xA9;      // 代码 © 符合于 to the '\xA9' 文字
   int d=0x263A;    // 代码 ☺ 符合于 the '\x263A' 文字
//--- 显示值
```

© 2000-2025, MetaQuotes Ltd.

```
   Print(a,b,c,d);
//--- 添加字符到字符串
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,b);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,c);
   Print(test);
//--- 成串的替换字符
   StringSetCharacter(test,0,d);
   Print(test);
//--- 合适的代码
   int a1=0x2660;
   int b1=0x2661;
   int c1=0x2662;
   int d1=0x2663;
//--- 添加黑桃字符
   StringSetCharacter(test,1,a1);
   Print(test);
//--- 添加红心字符
   StringSetCharacter(test,2,b1);
   Print(test);
//--- 添加方片字符
   StringSetCharacter(test,3,c1);
   Print(test);
//--- 添加梅花字符
   StringSetCharacter(test,4,d1);
   Print(test);
//--- 成串的字符文字示例
   test="Queen\x2660Ace\x2662";
   printf("%s",test);
  }
```

字符型的内部表示法是 无符号短整型 ，字符常量能够接受从0到65535的值。

另见

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

© 2000-2025, MetaQuotes Ltd.

# 日期时间型

日期时间型是为存储日期时间型预留的，开始日期是1970年1月1日，占8字节内存。

日期时间型常量可被当做数字串，由 6 个部分的字符组成：年、月、日 (或是日、月、年)、时、分、秒，数据以 D 开头，用单引号括起。

日期 (年、月、日)、时间 (时、分、秒)，或是一起被省略，值起于1970年1月1日，止于3000年12月31日。

根据日期的字面规定，您需要指定年，月和日。否则编译器会返回不完全输入的警告 。

示例：

```
datetime NY=D'2015.01.01 00:00';      // 2015年初的时间
datetime d1=D'1980.07.19 12:30:27';   // 年 月 日 小时 分钟 秒
datetime d2=D'19.07.1980 12:30:27';   // 等于 D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';         // 等于 D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';            // 等于 D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;              // 编译日期
datetime compilation_date_time=__DATETIME__;     // 编译日期和时间
datetime compilation_time=__DATETIME__-__DATE__;// 编译时间
//--- 返回编译器警告后的声明示例
datetime warning1=D'12:30:27';        // 等于 D'[编译日期] 12:30:27'
datetime warning2=D'';                // 等于 __DATETIME__
```

另见

日期类型结构, 日期和时间, 时间到字符串,, 字符串到时间

© 2000-2025, MetaQuotes Ltd.

# 颜色型

颜色型是为了存储颜色信息的，占用4个字节，头1个字节忽略不计，其他3个字节包括红绿蓝3个数据。

颜色数据可以用三种方法表示：字符数据、整型数据或者是颜色名（只能是已命名的 网页-颜色）。

字符数据的表达方法是用三个数字来表示三种主要颜色：红、绿、蓝的比例。数据以 C 开头，用单引号括住。数字的值在 0 ～ 255 之间按比例选取。

整数数据的表达方法使用十六进制或十进制数字。十六进制数字如 0x00BBGGRR，其中 RR 是红色元素的比例，GG 是绿色的比例，BB 是蓝色的比例。十进制数不能直接体现红绿蓝的比例，而是十六进制数字的十进制表示方式。

特殊颜色名可以参考 网页-颜色 表。

示例：

```
//--- 字面值
C'128,128,128'     // 灰色
C'0x00,0x00,0xFF' // 蓝色
//--- 颜色名称
clrRed                 // 红色
clrYellow              // 黄色
clrBlack               // 黑色
//--- 整数表示
0xFFFFFF           // 白色
16777215           // 白色
0x008000           // 绿色
32768              // 绿色
```

另见

网站色彩，颜色到字符串，字符串到颜色，类型转换

© 2000-2025, MetaQuotes Ltd.

# 布尔类型

布尔类型是用来存储true或者false的逻辑值的，它们的数字表示法分别是1和0。

示例：

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

内置法最多就是1字节，在逻辑表达式里可以注释，各种表达式都能用其他整数——编译器不会产生任何错误，在这种情况下，0值表示成false,其他值表示true。

示例：

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

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

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

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

//--- 执行结果
//    i= 5 且有真值
//    d= -2.5 且有真值
//    i= 0 且有错误值
//    d= 0 且有错误值
```

另见

布尔体系操作，优先规则

© 2000-2025, MetaQuotes Ltd.

# 枚举类型

枚举类型数据属于数据集合的限制额，最典型的是枚举类型：

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

该值列表是分割逗号命名的标识符常量列表。

示例：

```
enum months  // 已命名常量的计算
   {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
   };
```

计算清楚之后，结果是新的4字节整数数据类型值。新数据类型的描述可以严格地编译到通过的常量控制类型里，因为列举介绍了新命名的数据。在上述例子中，一月常量的值是0，二月是-1，十二月是-11。

规则：如果某一确定值并没有列举到命名常量-计算的一员，它的新值将会自动形成。如果是计算左边的，就会出现0值，在随后出现的值中，将会以先前计算的值为基准递增。

示例：

```
enum intervals  // 已命名常量的计算
   {
    month=1,     // 间隔一个月
    two_months,  // 两个月
    quarter,     // 三个月 - 四分之一
    halfyear=6,  // 半年
    year=12,     // 一年 - 12个月
   };
```

## 注释

- 不像C++，在MQL5里程序段的大小代表了计数类型，通常为4字节，也就是运算符 (月)的返回值是4。
- 不像C++，在MQL5里，匿名运算无法显示，在列举关键字后，需要指定独立名称。

© 2000-2025, MetaQuotes Ltd.

另见

类型转换

© 2000-2025, MetaQuotes Ltd.

# 真实型 (double, float)

真实型 (或浮点型)以小数部分为代表值，在MQL5语言里，浮点型数据有两种类型，在内存中实型数据的表示方法由 IEEE 754水平规定，它并不依赖平台、操作系统和程序语言。

| 类型 | 字节大小 | 最小正值 | 最大值 | C++ 类似物 |
|---|---|---|---|---|
| float | 4 | 1.175494351e-38 | 3.402823466e+38 | float |
| double | 8 | 2.2250738585072014e-308 | 1.7976931348623158e+308 | double |

## double

double (双精度)实数类型占用64位 (1个符号位，11个指数位，52个尾数位)。

## float

float (浮点)实数类型占用32位 (1个符号位，8个指数位，23个尾数位)。

## vector

double (双精度)类型数值的一维数组。动态分配数据内存。可以使用methods (方法)获得向量属性，同时可以更改向量大小。向量<double>条目可以用在模板函数。

## vectorf

如果精度亏损不严重，可以使用float (浮点)类型数值的一维数组来代替vector (向量)类型数值。向量<float>条目可以用在模板函数。

## vectorc

complex (复数)类型数值的一维数组旨在处理复数。向量<complex>条目可以用在模板函数。对vectorc类型向量的操作尚未执行。

## matrix

Matrix (矩阵)是double (双精度)类型数值的二维数组。动态分布矩阵元素的内存。可以使用methods (方法)获得矩阵属性，同时可以更改矩阵形状。矩阵<double>条目可以用在模板函数。

## matrixf

如果精度亏损不严重，可以使用float (浮点)类型数值的二维数组来代替matrix (矩阵)类型数值。矩阵<float>条目可以用在模板函数。

## matrixc

© 2000-2025, MetaQuotes Ltd.

complex (复数)类型数值的二维数组旨在处理复数。矩阵<complex>条目可以用在模板函数。对matrixc类型矩阵的操作尚未执行。

双精度名称是为了表示这些浮点型数据的双倍准确率，在大多数情况下，双精度型是最方便的，浮点型数据的精密度限制是不够的，原因就在于浮点型数据还要节约内存 (这就是真实型数据庞大的重要性)。

浮点型数据由整数部分、小数点(.)和小数部分组成，其中整数部分和小数部分为一系列十进制数字。

示例：

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

有更科学的方法输入实常数，通常这些方法比传统方法更简洁。

示例：

```
   double c1=1.12123515e-25;
   double c2=0.00000000000000000000000112123515; // 小数点后有24个零

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

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

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

在二进制系统中，实型数据以限制精确度来存储，而常用作十进制计数法。这就是在十进制系统中，许多被取代的数字在二进制系统中被输出为无数小数点的原因。

例如，0.3和0.7的小数部分被取代，而0.25却被精确保留，因为它的有效数字是两位。

就这一点而言，不要实际地区对比两个真实数据，因为对比是不精确的。

示例：

```
void OnStart()
  {
//---
   double three=3.0;
   double x,y,z;
   x=1/three;
   y=4/three;
   z=5/three;
   if(x+y==z)
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("1/3 + 4/3 == 5/3");
   else
      Print("1/3 + 4/3 != 5/3");
// 结果: 1/3 + 4/3 != 5/3
  }
```

如果你仍需要对比两个真实型数据，有两种方法，第一种，在同样的小数位对比他们之间的不同。

示例：

```
bool EqualDoubles(double d1,double d2,double epsilon)
  {
   if(epsilon<0) epsilon=-epsilon;
//---
   if(d1-d2>epsilon)
      return false;
   if(d1-d2<-epsilon)
      return false;
//---
   return true;
  }
void OnStart()
  {
   double d_val=0.7;
   float  f_val=0.7;
   if(EqualDoubles(d_val,f_val,0.000000000000001))
      Print(d_val," equals ",f_val);
   else
      Print("Different: d_val = ",DoubleToString(d_val,16),"  f_val = ",DoubleToStrinc
// 结果: 不同: d_val= 0.7000000000000000   f_val= 0.6999999880790710
  }
```

上例中第五位比DBL_EPSILON多，值是2.2204460492503131e-016，与浮点型数据相一致的是FLT_EPSILON = 1.192092896e-07。这些值有如下意义：满足条件的最低值 1.0 + DBL_EPSILON != 1.0（大量的浮点型数值 1.0 + FLT_EPSILON != 1.0）。

第二种方法通过0将两种真实型数据进行了标准对比，它是无意义的，因为任何标准化操作都能给出非标准的结果。

示例：

```
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0)
      return(true);
   else
      return(false);
  }
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   double d_val=0.3;
   float  f_val=0.3;
   if(CompareDoubles(d_val,f_val))
      Print(d_val," equals ",f_val);
   else
      Print("Different: d_val = ",DoubleToString(d_val,16),"  f_val = ",DoubleToStrinq
// 结果: 不同: d_val= 0.3000000000000000   f_val= 0.3000000119209290
  }
```

一些数字协同处理器的操作能够导致无效的真实型数字，不能运用到数字操作和对比中，因为用无效真实型数据的操作结果是不能定义的。例如，当想要计算2的反正弦，结果可能无穷负。

示例：

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

除了负无穷大也还有正无穷大和NaN（不是数字），确定数字是否是无效的，可以运用MathIsValidNumber()。功能，根据IEEE标准，可以用专用机描述。例如，双精度型正无穷代表小的0x7FF0 0000 0000 0000。

示例：

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

//--- 开始
   str1 s1;
   str2 s2;
//---
   s1.d=MathArcsin(2.0);         // 获得无效数据 -1.#IND
   s2=s1;
   printf("1.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFFF000000000000;      // 无效数据 -1.#QNAN
   s1=s2;
   printf("2.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF7000000000000;      // 最大 nonnumber SNaN
   s1=s2;
   printf("3.   %f %I64X",s1.d,s2.l);
//---
```

© 2000-2025, MetaQuotes Ltd.

```
   s2.l=0x7FF8000000000000;      // 最小 nonnumber QNaN
   s1=s2;
   printf("4.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FFF000000000000;      // 最大 nonnumber QNaN
   s1=s2;
   printf("5.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF0000000000000;      // 正无穷大 1.#INF 和最小 nnonnumber SNaN
   s1=s2;
   printf("6.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFF0000000000000;      // 负无穷大 -1.#INF
   s1=s2;
   printf("7.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x8000000000000000;      // 负零 -0.0
   s1=s2;
   printf("8.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FE0000000000000;      // 0.5
   s1=s2;
   printf("9.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FF0000000000000;      // 1.0
   s1=s2;
   printf("10.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FEFFFFFFFFFFFFF;      // 最大的规格化数字 (MAX_DBL)
   s1=s2;
   printf("11.  %.16e %I64X",s1.d,s2.l);
//---
   s2.l=0x0010000000000000;      // 最小的正规格化 (MIN_DBL)
   s1=s2;
   printf("12.  %.16e %.16I64X",s1.d,s2.l);
//---
   s1.d=0.7;                     // 显示数字0.7-无限循环部分
   s2=s1;
   printf("13.  %.16e %.16I64X",s1.d,s2.l);
/*
1.  -1.#IND00 FFF8000000000000
2.  -1.#QNAN0 FFFF000000000000
3.   1.#SNAN0 7FF7000000000000
4.   1.#QNAN0 7FF8000000000000
5.   1.#QNAN0 7FFF000000000000
6.   1.#INF00 7FF0000000000000
7.  -1.#INF00 FFF0000000000000
8.  -0.000000 8000000000000000
9.   0.500000 3FE0000000000000
```

© 2000-2025, MetaQuotes Ltd.

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

另见

双精度型到字符串，标准化双精度型，数字类型常量

© 2000-2025, MetaQuotes Ltd.

# 复数（complex）

内置的complex类型是具有两个double字段的结构：

```
struct complex
  {
   double            real;    // 实数部分
   double            imag;    // 虚数部分
  };
```

"复数"类型可以作为MQL5函数的参数按值传递(与普通结构不同，普通结构只通过引用传递)。对于从DLL导入的函数，"复数"类型只能通过引用来传递。

'i'后缀用于描述复常数：

```
complex square(complex c)
  {
   return(c*c);
  }
void OnStart()
  {
   Print(square(1+2i));  // 常量作为参数传递
  }
// "(-3,4)"将被输出，这是复数的字符串表示
```

目前，复数只有简单的运算可用：=、+、-、*、/、+=、-=、*=、/=、==、!=。

之后将添加对附加数学函数的支持，以支持计算绝对值、正弦、余弦等。

## vectorc

complex（复数）类型数值的一维数组旨在处理复数。向量<complex>条目可以用在模板函数。对vectorc类型向量的操作尚未执行。

## matrix

Matrix（矩阵）是double（双精度）类型数值的二维数组。动态分布矩阵元素的内存。可以使用methods（方法）获得矩阵属性，同时可以更改矩阵形状。矩阵<double>条目可以用在模板函数。

## matrixf

如果精度亏损不严重，可以使用float（浮点）类型数值的二维数组来代替matrix（矩阵）类型数值。矩阵<float>条目可以用在模板函数。

## matrixc

complex（复数）类型数值的二维数组旨在处理复数。矩阵<complex>条目可以用在模板函数。对matrixc类型矩阵的操作尚未执行。

© 2000-2025, MetaQuotes Ltd.

# 字符串数据

字符串数据是用来存储文本串的，文本串是以编码的形式末尾为0的字符序列，每串常数分配给一个变量，每串数据都用双引号引出写着“这是一串字符常量”。

如果字符串中包括双引号(")，那么反斜杠字符(\)必须放其前面。如果反斜杠字符放其前面的话，任何特殊字符常量都可以写入字符串。

示例：

```
string svar="This is a character string";
string svar2=StringSubstr(svar,0,4);
Print("Copyright symbol\t\x00A9");
FileWrite(handle,"This string contains a new line symbols \n");
string MT5path="C:\\Program Files\\MetaTrader 5";
```

使源代码具有可读性，长型常量字符串无需额外操作即可分成几个部分。编译期间，这些部分将会合并成一个长字符串：

```
//--- 声明长型常量字符串
   string HTML_head="<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\""
                    " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n"
                    "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
                    "<head>\n"
                    "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ut
                    "<title>Trade Operations Report</title>\n"
                    "</head>";
//--- 导出常量字符串到日志
   Print(HTML_head);
  }
```

另见

转换函数，字符串函数，打开文件，文件阅读字符串，文件输入字符串

© 2000-2025, MetaQuotes Ltd.

# 架构和类以及界面

## 架构

架构是设定任何类型的元素（除了 空 型），因此，架构要组合不同类型的逻辑相关资料。

### 架构说明

以下描述定义结构类型数据：

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

架构名称不能用作标识符（变量或功能的名称），应该直接记录在MQL5架构元素中，没有定位，在C++中需要遵循以下指令来定义这些命令：

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

如果想在架构中执行另一定位，运用辅助程序，"fillers"填充正确大小。

示例：

```
struct trade_settings
  {
   uchar  slippage;     // 许可的下降值-1字节大小
   char   reserved1;    // 跳过 1 字节
   short  reserved2;    // 跳过 2 字节
   int    reserved4;    // 跳过另4个字节。确保定位8字节界限
   double take;         // 固定利润价格值
   double stop;         // 受保护的止损价格值
  };
```

线性构造的描述只对动态传输功能入口有效。

**注意**：该例子列举的是错误的设计数据，在双精度型数据中，首先声明*take*和*stop*的数据大小比较好，然后是*slippage*，在此情况下，数据的内部表达式与 *#pragma pack()* 无关，都是一样的。

如果该构造包含字符串型数据变量，和/或目标动态数组，编译器给架构分派隐含的构造函数。构造函数设置所有字符串型和初始化目标动态数组的所有结构。

### 简单构造

没有包含字符串，类对象，指针和动态数组对象的构造被称为简单构造。简单构造的变量及其数组可以作为参数从DLL被传递到imported函数。

仅允许在下面两种情况下复制简单构造：

- 如果对象属于相同的构造类型
- 如果对象通过血统连接，这意味着一个构造是另一个构造的后代。

© 2000-2025, MetaQuotes Ltd.

若要提供示例，让我们来开发一个CustomMqlTick自定义构造，其内容与内置的MqlTick内容是一致的。编译器不允许复制MqlTick对象值到CustomMqlTick类型对象。直接类型转换到所需类型也会引起编译错误：

```
      //--- 禁止复制不同类型的简单构造
      my_tick1=last_tick;                // 这里编译器返回错误

      //--- 同时也禁止不同类型构造彼此的类型转换
      my_tick1=(CustomMqlTick)last_tick;// 这里编译器返回错误
```

因此，只留有一个选项 -依次复制构造元素的值。仍然允许复制CustomMqlTick相同类型的值。

```
      CustomMqlTick my_tick1,my_tick2;
      //--- 允许复制CustomMqlTick以下相同类型的对象
      my_tick2=my_tick1;

      //--- 从CustomMqlTick简单构造对象创建一个数组并为其写值
      CustomMqlTick arr[2];
      arr[0]=my_tick1;
      arr[1]=my_tick2;
```

调用ArrayPrint()函数进行检查以在日志中显示*arr[]数组值*。

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 开发类似于内置MqlTick的构造
   struct CustomMqlTick
     {
      datetime          time;          // 最后价格更新时间
      double            bid;           // 当前卖价
      double            ask;           // 当前买价
      double            last;          // 最后一笔交易的当前价格(Last)
      ulong             volume;        // 当前最后价格的交易量
      long              time_msc;      // 以毫秒计算的最后价格更新时间
      uint              flags;         // 报价标识
     };
   //--- 获得最后报价值
   MqlTick last_tick;
   CustomMqlTick my_tick1,my_tick2;
//--- 尝试从MqlTick复制数据到CustomMqlTick
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      //--- 禁止复制不相关的简单构造
      //1. my_tick1=last_tick;               // 这里编译器返回错误

      //--- 同时也禁止不相关构造彼此的类型转换
      //2. my_tick1=(CustomMqlTick)last_tick;// 这里编译器返回错误
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 因此，依次复制构造成员
      my_tick1.time=last_tick.time;
      my_tick1.bid=last_tick.bid;
      my_tick1.ask=last_tick.ask;
      my_tick1.volume=last_tick.volume;
      my_tick1.time_msc=last_tick.time_msc;
      my_tick1.flags=last_tick.flags;

      //--- 允许复制CustomMqlTick以下相同类型的对象
      my_tick2=my_tick1;

      //--- 从CustomMqlTick简单构造对象创建一个数组并为其写值
      CustomMqlTick arr[2];
      arr[0]=my_tick1;
      arr[1]=my_tick2;
      ArrayPrint(arr);
//--- 显示包含CustomMqlTick类型对象的数组值的示例
      /*
                        [time]    [bid]    [ask]    [last] [volume]     [time_msc] [flags]
      [0] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000  1450000 1496070277157       2
      [1] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000  1450000 1496070277157       2
      */
     }
   else
      Print("SymbolInfoTick() failed, error = ",GetLastError());
  }
```

第二个示例显示通过血统复制简单构造的特性。假设我们有Animal基本构造，从此得到了Cat和Dog的构造。我们可以彼此复制Anima和Cat对象，以及Anima和Dog对象，但我们不能彼此复制Cat和Dog，虽然这两种都是衍生自Anima构造。

```
//--- 描述狗的构造
struct Dog: Animal
  {
   bool              hunting;        // 狩猎犬
  };
//--- 描述猫的构造
struct Cat: Animal
  {
   bool              home;           // 家猫
  };
//--- 创建子构造对象
   Dog dog;
   Cat cat;
//--- 可以从母系复制到后代 (Animal ==> Dog)
   dog=some_animal;
   dog.swim=true;     // 狗可以游泳
//--- 您不能复制子构造的对象 (Dog != Cat)
```

© 2000-2025, MetaQuotes Ltd.

```
   cat=dog;          // 编译器返回错误
```

完整的示例代码：

```
//--- 描述动物的基本构造
struct Animal
  {
   int               head;          // 头部数量
   int               legs;          // 腿的数量
   int               wings;         // 翅膀数量
   bool              tail;          // 尾巴
   bool              fly;           // 飞行
   bool              swim;          // 游泳
   bool              run;           // 跑步
  };
//--- 描述狗的构造
struct Dog: Animal
  {
   bool              hunting;       // 狩猎犬
  };
//--- 描述猫的构造
struct Cat: Animal
  {
   bool              home;          // 家猫
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建和描述基本Animal类型的对象
   Animal some_animal;
   some_animal.head=1;
   some_animal.legs=4;
   some_animal.wings=0;
   some_animal.tail=true;
   some_animal.fly=false;
   some_animal.swim=false;
   some_animal.run=true;
//--- 创建子类型对象
   Dog dog;
   Cat cat;
//--- 可以从母系复制到后代 (Animal ==> Dog)
   dog=some_animal;
   dog.swim=true;    // 狗可以游泳
//--- 您不能复制子构造的对象 (Dog != Cat)
   //cat=dog;          // 这里编译器返回错误
//--- 因此，只可以依次复制元素
   cat.head=dog.head;
```

© 2000-2025, MetaQuotes Ltd.

```
   cat.legs=dog.legs;
   cat.wings=dog.wings;
   cat.tail=dog.tail;
   cat.fly=dog.fly;
   cat.swim=false;   // 猫不可以游泳
//--- 可以从后代复制值到祖先
   Animal elephant;
   elephant=cat;
   elephant.run=false;// 大象不可以跑步
   elephant.swim=true;// 大象可以游泳
//--- 创建数组
   Animal animals[4];
   animals[0]=some_animal;
   animals[1]=dog;
   animals[2]=cat;
   animals[3]=elephant;
//--- 打印出
   ArrayPrint(animals);
//--- 执行结果
/*
        [head] [legs] [wings] [tail] [fly] [swim] [run]
   [0]      1      4        0   true false  false  true
   [1]      1      4        0   true false   true  true
   [2]      1      4        0   true false  false false
   [3]      1      4        0   true false   true false
*/
  }
```

复制简单类型的另一种方式是使用一个联合。构造对象应该是相同联合的成员 ——请见union中的示例。

## 接入架构会员入口

架构是一种新的数据型用来声明这种类型变量的，只有在一个项目开始是架构可以运行，架构会员可以通过 操作点 (.)进入接口。

示例：

```
struct trade_settings
  {
   double take;         // 利润固定价格值
   double stop;         // 受保护的止损价格值
   uchar  slippage;     // 可接受的下降值
  };
//--- 创建和初始化交易设置类型的变量
trade_settings my_set={0.0,0.0,5};
if (input_TP>0) my_set.take=input_TP;
```

## 'pack'用于对齐架构和类字段

© 2000-2025, MetaQuotes Ltd.

特殊的pack属性可以设置架构或类字段的对齐。

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

这里n是以下值中的一个：1，2，4，8或16。它也可以不存在。

例如：

```
   struct pack(sizeof(long)) MyStruct
     {
      //构造成员要对齐到8字节分界线
     };
或者
   struct MyStruct pack(sizeof(long))
     {
      //构造成员要对齐到8字节分界线
     };
```

'pack(1)'被默认应用于架构。这意味着构造成员陆续定位在内存中，架构大小等于成员大小的总和。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 架构简单，无对齐
   struct Simple_Structure
     {
      char              c; // sizeof(char)=1
      short             s; // sizeof(short)=2
      int               i; // sizeof(int)=4
      double            d; // sizeof(double)=8
     };
   //--- 声明一个简单的架构实例
   Simple_Structure s;
//--- 显示每个构造成员的大小
   Print("sizeof(s.c)=",sizeof(s.c));
   Print("sizeof(s.s)=",sizeof(s.s));
   Print("sizeof(s.i)=",sizeof(s.i));
   Print("sizeof(s.d)=",sizeof(s.d));
//--- 确保POD架构的大小等于其成员大小之和
   Print("sizeof(simple_structure)=",sizeof(simple_structure));
/*
  结果：
   sizeof(s.c)=1
   sizeof(s.s)=2
   sizeof(s.i)=4
   sizeof(s.d)=8
```

© 2000-2025, MetaQuotes Ltd.

```
   sizeof(simple_structure)=15
*/
  }
```

当与应用这种对齐的第三方程序库(*.DLL)交换数据时，可能需要对架构字段进行对齐。

让我们使用一些示例来显示对齐的工作方式。我们将应用由四个没有对齐的成员组成的架构。

```
//--- 架构简单，无对齐
   struct Simple_Structure pack() // 没有指定大小，对齐设置距离边界线1字节
     {
      char              c; // sizeof(char)=1
      short             s; // sizeof(short)=2
      int               i; // sizeof(int)=4
      double            d; // sizeof(double)=8
     };
//--- 声明一个简单的架构实例
   Simple_Structure s;
```

架构字段将根据声明的顺序和类型大小陆续定位在内存中. 架构大小为15，而数组中架构字段的偏移量未定义。

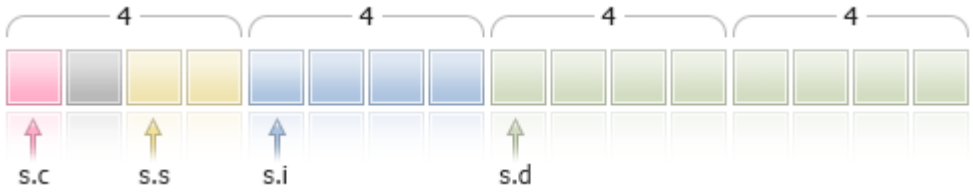
现在声明与4字节对齐方式相同的架构并运行代码。

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 架构简单，4字节对齐
   struct Simple_Structure pack(4)
     {
      char              c; // sizeof(char)=1
      short             s; // sizeof(short)=2
      int               i; // sizeof(int)=4
      double            d; // sizeof(double)=8
     };
   //--- 声明一个简单的架构实例
   Simple_Structure s;
//--- 显示每个构造成员的大小
   Print("sizeof(s.c)=",sizeof(s.c));
   Print("sizeof(s.s)=",sizeof(s.s));
   Print("sizeof(s.i)=",sizeof(s.i));
   Print("sizeof(s.d)=",sizeof(s.d));
//--- 确保POD架构的大小现在不等于其成员大小之和
   Print("sizeof(simple_structure)=",sizeof(simple_structure));
```

© 2000-2025, MetaQuotes Ltd.

```
/*
  结果:
   sizeof(s.c)=1
   sizeof(s.s)=2
   sizeof(s.i)=4
   sizeof(s.d)=8
   sizeof(simple_structure)=16 // 架构大小已更改
*/
  }
```

该架构大小已更改，因此所有4字节及以上的成员从架构开始就有一个4字节倍数的偏移量。较小的成员将与其自身大小的边界线对齐（例如，2用于'short'）。这就是它的外观（添加的字节以灰色显示）。




在这种情况下，**s.c**成员添加1字节，以便**s.s**（sizeof(short)==2)字段的边界线为2字节（对齐用于'short'类型）。

数组中架构开始的偏移量也对齐到4字节，即对于Simple_Structure arr[]来说，元素a[0]、a[1]和a[n]的地址也将是4字节的倍数。

让我们再考虑另外两个由类似类型组成的架构，即4字节对齐但不同的成员顺序。在第一个架构中，成员按类型大小升序排列。

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 与4字节边界线对齐的简单架构
   struct CharShortInt pack(4)
     {
      char              c; // sizeof(char)=1
      short             s; // sizeof(short)=2
      int               i; // sizeof(double)=4
     };
//--- 声明一个简单的架构实例
   CharShortInt ch_sh_in;
//--- 显示每个构造成员的大小
   Print("sizeof(ch_sh_in.c)=",sizeof(ch_sh_in.c));
   Print("sizeof(ch_sh_in.s)=",sizeof(ch_sh_in.s));
   Print("sizeof(ch_sh_in.i)=",sizeof(ch_sh_in.i));

//--- 确保POD架构的大小等于其成员大小之和
   Print("sizeof(CharShortInt)=",sizeof(CharShortInt));
```

© 2000-2025, MetaQuotes Ltd.

```
/*
  结 果 :
   sizeof(ch_sh_in.c)=1
   sizeof(ch_sh_in.s)=2
   sizeof(ch_sh_in.i)=4
   sizeof(CharShortInt)=8
*/
  }
```

正如我们所见，架构大小为8，由两个4字节区块组成。第一个区块包含'char'和'short'类型的字段，而第二个区块包括'int'类型的字段。

现在，让我们把第一个架构变为第二个架构，只是字段顺序不同，即将'short'类型成员移动到最后。

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 与4字节边界线对齐的简单架构
   struct CharIntShort pack(4)
     {
      char              c; // sizeof(char)=1
      int               i; // sizeof(double)=4
      short             s; // sizeof(short)=2
     };
//--- 声明一个简单的架构实例
   CharIntShort ch_in_sh;
//--- 显示每个构造成员的大小
   Print("sizeof(ch_in_sh.c)=",sizeof(ch_in_sh.c));
   Print("sizeof(ch_in_sh.i)=",sizeof(ch_in_sh.i));
   Print("sizeof(ch_in_sh.s)=",sizeof(ch_in_sh.s));
//--- 确保POD架构的大小等于其成员大小之和
   Print("sizeof(CharIntShort)=",sizeof(CharIntShort));
/*
  结 果 :
   sizeof(ch_in_sh.c)=1
   sizeof(ch_in_sh.i)=4
   sizeof(ch_in_sh.s)=2
   sizeof(CharIntShort)=12
*/
  }
```

虽然架构内容没有更改，但是改变成员序列增加了架构的大小。

© 2000-2025, MetaQuotes Ltd.

在继承时还应该考虑对齐。让我们使用具有单个'char'类型成员的简单 "父"架构来演示这一点。没有对齐的架构大小为1。

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

让我们创建具有'short' (sizeof(short)=2)类型成员的 "子"类。

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

因此，当对齐设置为2字节时，虽然成员大小为3，但架构大小等于4。在本示例中，将 "父"类分配2字节，以便 "子"类'short'字段的访问对齐到2字节。

如果MQL5应用程序通过在文件或文件流级别上的编写/读取与第三方数据互动，则需要了解内存如何分配到构造成员。

标准程序库的MQL5\Include\WinAPI目录包含使用WinAPI函数的函数。当需要使用WinAPI时，这些函数应用具有指定对齐方式的架构。

offsetof是一个与 pack属性直接相关的特殊命令。它允许我们从架构的开始获取成员的偏移量。

```
//--- 声明 "子"类型变量。
   Children child;
//--- 从架构的开始检测偏移量
   Print("offsetof(Children,c)=",offsetof(Children,c));
   Print("offsetof(Children,s)=",offsetof(Children,s));
/*
  结果：
   offsetof(Children,c)=0
   offsetof(Children,s)=2
*/
```

## 修饰符 'final'

架构声明过程中使用'final' 修饰符禁止从该架构进一步继承。如果架构无需进一步修改，或出于安全原因不允许修改，则以'final'修饰符声明该架构。此外，该架构的所有成员也将默认为不可更改。

© 2000-2025, MetaQuotes Ltd.

```
struct settings final
  {
  //--- 构造主体
  };

struct trade_settings : public settings
  {
  //--- 构造主体
  };
```

如果您像上面示例一样试图以'final'修饰符继承形成一个架构，编译器将返回一个错误：

```
不能像其被声明为'final'一样从'设置'继承
参照'设置'声明
```

## 类

类与架构的不同之处在于：

- 在声明中使用关键字类;
- 默认情况下，所有类成员都能通过独自通过指示符，除非另行表明的，架构里的数据成员都能通过默认类型，除非是另行标注的。
- 甚至在类中没有虚函数的情况下，类目标总是有一个 虚函数 图表，结构类没有虚函数;
- 新功能操作也能应用到类目标中，但不能应用到结构类中;
- 类只能从类中 继承，结构也只能从架构中继承。

类和架构都有明确的构造函数和解构方法，如果构造函数定义明确，架构或类别变量的初始化进行初始化序列是不可能的。

示例：

```
struct trade_settings
  {
   double take;          // 利润固定价格值
   double stop;          // 受保护的止损价格值
   uchar  slippage;      // 可接受的下降值
   //--- 构造函数
          trade_settings() { take=0.0; stop=0.0; slippage=5; }
   //--- 析构函数
         ~trade_settings() { Print("This is the end"); }
  };
//--- 编译器生成一个无法初始化的错误信息
trade_settings my_set={0.0,0.0,5};
```

### 构造函数和析构函数

构造函数是一种特殊函数，当创建结构或类的对象时自动调用并且通常用来初始化 类成员。从更长远来看我们将只看重类，并且同样应用于架构中，除非是另行定义。构造函数的名称要与类函数的名称匹配。构造函数不需要返回类型（您可以指定空型）。

定义类成员 -字符串，动态数组 和需要初始化的对象 -无论如何都需要初始化，不管构造函数是否存在。

© 2000-2025, MetaQuotes Ltd.

每个类可以有多个构造函数，根据参数数量和初始化列表而有所不同。需要指定参数的构造函数称为参数构造函数。

无参数构造函数被称为**缺省构造函数**。如果一个类中没有声明构造函数，编译器会在编译过程中创建一个缺省构造函数。

```
//+------------------------------------------------------------------+
//| 处理日期的类                                                     |
//+------------------------------------------------------------------+
class MyDateClass
  {
private:
   int               m_year;           // 年
   int               m_month;          // 月
   int               m_day;            // 几月几日
   int               m_hour;           // 某天几时
   int               m_minute;         // 分钟
   int               m_second;         // 秒
public:
   //--- 缺省构造函数
                     MyDateClass(void);
   //--- 参数构造函数
                     MyDateClass(int h,int m,int s);
  };
```

构造函数可以在类描述中声明，然后定义主体。例如，MyDateClass的两个构造函数可以定义如下：

```
//+------------------------------------------------------------------+
//| 默认构造函数                                                     |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(void)
  {
//---
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=mdt.hour;
   m_minute=mdt.min;
   m_second=mdt.sec;
   Print(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| 参数构造函数                                                     |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(int h,int m,int s)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=h;
   m_minute=m;
   m_second=s;
   Print(__FUNCTION__);
  }
```

在 默认构造函数，类的所有成员都使用TimeCurrent() 函数，在参数构造函数，只有小时值在使用。其他的类成员 (m_year, m_month 和 m_day) 将自动初始化当前日期。

当初始化类的对象数组时，默认构造函数有一个特殊用途。所有参数都有默认值的构造函数，并不是默认构造函数。示例如下：

```
//+------------------------------------------------------------------+
//| 默认构造函数的类                                                 |
//+------------------------------------------------------------------+
class CFoo
  {
   datetime          m_call_time;      // 最近一次对象调用时间
public:
   //--- 带有默认值参数的构造函数不是默认构造函数
                     CFoo(const datetime t=0){m_call_time=t;};
   //--- 复制构造函数
                     CFoo(const CFoo &foo){m_call_time=foo.m_call_time;};

   string ToString(){return(TimeToString(m_call_time,TIME_DATE|TIME_SECONDS));};
  };
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
// CFoo foo; // 该变量不能使用 - 没有设置默认构造函数
//--- 创建 CFoo 对象的可能选项
   CFoo foo1(TimeCurrent());      // 参数构造函数的显式调用
   CFoo foo2();                   // 带有默认参数的参数构造函数的显式调用
   CFoo foo3=D'2009.09.09';       // 参数构造函数的隐式调用
   CFoo foo40(foo1);              // 复制构造函数的显式调用
   CFoo foo41=foo1;               // 复制构造函数的隐式调用
   CFoo foo5;                     // 默认构造函数的显式调用 如果没有默认构造函数,
                                  // 那么带有默认值的参数构造函数被调用)
//--- 接收 CFoo 指针的可能选项
   CFoo *pfoo6=new CFoo();        // 动态创建对象和接收其指针
   CFoo *pfoo7=new CFoo(TimeCurrent());// 另一个动态对象创建的选项
   CFoo *pfoo8=GetPointer(foo1); // 现在 pfoo8 指向对象 foo1
```

© 2000-2025, MetaQuotes Ltd.

```
   CFoo *pfoo9=pfoo7;             // pfoo9 和 pfoo7 指向一个和相同的对象
   // CFoo foo_array[3];          // 该选项不能使用 - 没有指定默认构造函数
//--- 显示m_call_time值
   Print("foo1.m_call_time=",foo1.ToString());
   Print("foo2.m_call_time=",foo2.ToString());
   Print("foo3.m_call_time=",foo3.ToString());
   Print("foo4.m_call_time=",foo4.ToString());
   Print("foo5.m_call_time=",foo5.ToString());
   Print("pfoo6.m_call_time=",pfoo6.ToString());
   Print("pfoo7.m_call_time=",pfoo7.ToString());
   Print("pfoo8.m_call_time=",pfoo8.ToString());
   Print("pfoo9.m_call_time=",pfoo9.ToString());
//--- 删除动态创建数组
   delete pfoo6;
   delete pfoo7;
   //删除 pfoo8;  // 您不需要一定删除pfoo8，因为它指向自动创建的对象foo1
   //删除 pfoo9;  // 您不需要一定删除pfoo9，因为它指向pfoo7相同的对象
  }
```

如果您取消这些字符串

```
  //CFoo foo_array[3];     // 该变量不能使用 - 没有设置默认构造函数
```

或

```
  //CFoo foo_dyn_array[];  // 该变量不能使用 - 没有设置默认构造函数
```

然后编译器将会返回一个错误 "默认构造函数未定义"。

如果类有用户定义构造函数，编译器就不会生成默认构造函数。这意味着如果一个类中声明参数构造函数，但未声明默认构造函数，则您不能声明类对象的数组。编译器将返回这个脚本错误：

```
//+------------------------------------------------------------------+
//| 无默认构造函数的类                                               |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
                     CFoo(string name) { m_name=name;}
  };
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 编译过程中收到 "默认构造函数未定义"的错误
   CFoo badFoo[5];
  }
```

© 2000-2025, MetaQuotes Ltd.

在该示例中，CFoo 类拥有声明的参数构造函数 - 在这种情况下，编译器在编译过程中不能自动创建默认构造函数。同时当您声明对象数组时，假设所有对象应该自动创建和初始化。对象自动初始化过程中，需要调用默认构造函数，但由于默认构造函数不是显式声明并且不会通过编译器自动生成，所以不可能创建这种对象。因此，编译器在编译阶段会生成一个错误。

有一个使用构造函数初始化对象的特殊语法。结构和类成员的构造函数初始化软件（特殊结构初始化）可以在初始化列表中指定。

初始化列表就是通过逗号隔开的初始化软件列表，它在构造函数 参数列表 后主体 前（左大括号前）的冒号后面。它有以下几点要求：

- 初始化列表仅能用于 构造函数;
- 父成员 不能在初始化列表中初始化；
- 初始化列表必须遵循一个函数 定义 （实施）。

这是一个用于初始化类成员的几个构造函数的示例。

```
//+------------------------------------------------------------------+
//| 存储字符名称的类                                                 |
//+------------------------------------------------------------------+
class CPerson
  {
   string            m_first_name;     // 第一名称
   string            m_second_name;    // 第二名称
public:
   //--- 空默认构造函数
                     CPerson() {Print(__FUNCTION__);};
   //--- 参数构造函数
                     CPerson(string full_name);
   //--- 初始化列表的构造函数
                     CPerson(string surname,string name): m_second_name(surname), m_fi
   void PrintName(){PrintFormat("Name=%s Surname=%s",m_first_name,m_second_name);};
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
CPerson::CPerson(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>=0)
     {
      m_first_name=StringSubstr(full_name,0,pos);
      m_second_name=StringSubstr(full_name,pos+1);
     }
  }
//+------------------------------------------------------------------+
//|脚本程序开始函数                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 收到 "默认构造函数未定义"的错误
   CPerson people[5];
   CPerson Tom="Tom Sawyer";                          // 汤姆 索亚
   CPerson Huck("Huckleberry","Finn");                // 哈克贝利 费恩
   CPerson *Pooh = new CPerson("Winnie","Pooh");      // 维尼熊
   //--- 输出值
   Tom.PrintName();
   Huck.PrintName();
   Pooh.PrintName();

   //--- 删除一个动态创建的对象
   delete Pooh;
  }
```

在这种情况下，CPerson 类有三种构造函数：

1. 显式 默认构造函数，允许创建该类对象的数组；
2. 单参数构造函数，会得到一个作为参数的完整名称并根据发现的空间分成名称和第二名称；
3. 双参数的构造函数包含 初始化列表。初始化软件 - m_second_name 姓)和 m_first_name (名)。

注意使用列表初始化已经替代了一个任务。个体成员必须初始化为：

```
 class_member (表达式列表)
```

在初始化列表，成员可按任意顺序排列，但所有的类成员将会根据它们公告的顺序初始化。这意味着在第三构造函数，第一个m_first_name成员将会初始化，因为它是最先公告的，并且仅在m_second_name初始化之后。如果一些类成员的初始化取决于其他类成员的值，则应该将此考虑其中。

如果默认构造函数没有在基本类声明，而同时声明一个或多个参数函数，您应该保持调用初始化列表中的一个基本类的构造函数。它作为列表普通成员用逗号分隔并且无论初始化列表位于哪里，都在对象初始化时被最先调用。

```
//+------------------------------------------------------------------+
//| 基本类                                                           |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
   //--- 初始化列表的构造函数
                     CFoo(string name) : m_name(name) { Print(m_name);}
  };
//+------------------------------------------------------------------+
//| 派生自CFoo 的类                                                  |
//+------------------------------------------------------------------+
class CBar : CFoo
  {
   CFoo              m_member;       // 类成员是父对象
```

© 2000-2025, MetaQuotes Ltd.

```
public:
   //--- 初始化列表中的默认构造函数调用父构造函数
                     CBar(): m_member(_Symbol), CFoo("CBAR") {Print(__FUNCTION__);}
  };
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   CBar bar;
  }
```

例如，创建柱对象时，将会调用默认构造函数CBar()，这里首先调用父CFoo构造函数，然后是m_member类成员的构造函数。

析构函数是一种特殊功能，当类目标被破坏时自动调用，析构函数的名称用波浪字符(~)以分类名输入。串型数据、动态数组和目标函数，不管破坏函数是否出现，无论如何都不会初始化。如果存在破坏函数，该行为在召回破坏者后会执行。

破坏函数总是 虚拟的 ，无论虚拟值存在与否。

## 规定的分类方法

分类功能方式既能被内在分类定义也能被外在分类定义，如果该方法被分类定义，主题会正确显示。

示例：

© 2000-2025, MetaQuotes Ltd.

```
class CTetrisShape
  {
protected:
   int               m_type;
   int               m_xpos;
   int               m_ypos;
   int               m_xsize;
   int               m_ysize;
   int               m_prev_turn;
   int               m_turn;
   int               m_right_border;
public:
   void              CTetrisShape();
   void              SetRightBorder(int border) { m_right_border=border; }
   void              SetYPos(int ypos)          { m_ypos=ypos;           }
   void              SetXPos(int xpos)          { m_xpos=xpos;           }
   int               GetYPos()                  { return(m_ypos);        }
   int               GetXPos()                  { return(m_xpos);        }
   int               GetYSize()                 { return(m_ysize);       }
   int               GetXSize()                 { return(m_xsize);       }
   int               GetType()                  { return(m_type);        }
   void              Left()                     { m_xpos-=SHAPE_SIZE;    }
   void              Right()                    { m_xpos+=SHAPE_SIZE;    }
   void              Rotate()                   { m_prev_turn=m_turn; if(++m_turn>3) r
   virtual void      Draw()                     { return;                }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

SetRightBorder（整数边框）功能直接被内置的CTetrisShape定义。

CTetrisShape 构造函数和CheckDown(int& pad_array[]), CheckLeft(int& side_row[]) 、CheckRight(int& side_row[]) 方法在未被定义的情况下只能被内置分类定义，这些功能的定义会在代码中进一步体现。为了定义外在分类方法，使用范围解析操作功能，该分类名称与范围名称一样使用。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 基本类的构造函数                                                 |
//+------------------------------------------------------------------+
void CTetrisShape::CTetrisShape()
  {
   m_type=0;
   m_ypos=0;
   m_xpos=0;
   m_xsize=SHAPE_SIZE;
   m_ysize=SHAPE_SIZE;
   m_prev_turn=0;
   m_turn=0;
   m_right_border=0;
  }
//+------------------------------------------------------------------+
//| 检测向下的能力 (为竖条和方块)                                    |
//+------------------------------------------------------------------+
bool CTetrisShape::CheckDown(int& pad_array[])
  {
   int i,xsize=m_xsize/SHAPE_SIZE;
//---
   for(i=0; i<xsize; i++)
     {
      if(m_ypos+m_ysize>=pad_array[i]) return(false);
     }
//---
   return(true);
  }
```

## Public, Protected and Private Access Specifiers

当一个新的类进行时，会从外部限制类进入，关键字private 或者 protected此时就起到了作用，在此情况下，隐藏数据只能从相同分类的功能方法通路计入，如果保护关键字使用了，隐藏数据只能从分类方法中接入，就是该分类的继承接入，限制接入到分类方法中时也可以使用相同方法。

如果需要完全打开此链接，使用public关键字。

示例：

```
class CTetrisField
  {
private:
   int               m_score;                                // 得分
   int               m_ypos;                                 // 图形当前位置
   int               m_field[FIELD_HEIGHT][FIELD_WIDTH]; //  DOM矩阵
   int               m_rows[FIELD_HEIGHT];                //  DOM 行的编号
   int               m_last_row;                             // 最后一个自由行
   CTetrisShape     *m_shape;                                // 俄罗斯方块图形
   bool              m_bover;                                // 游戏结束
public:
   void              CTetrisField() { m_shape=NULL; m_bover=false; }
   void              Init();
   void              Deinit();
   void              Down();
   void              Left();
```

© 2000-2025, MetaQuotes Ltd.

```
   void              Right();
   void              Rotate();
   void              Drop();
private:
   void              NewShape();
   void              CheckAndDeleteRows();
   void              LabelOver();
  };
```

任何类成员和方法都在公开说明符后声明：(在下一个指示符输入之前)可以通过程序计入目标，示例中需要如下按键功能：CTetrisField(), Init(), Deinit(), Down(), Left(), Right(), Rotate() 和 Drop()。

在接入到说明符元素后，每个成员都要自动声明：(在接入下一个说明符之前)只能通过该类成员功能接入，接入到元素的分类通常在冒号(:)后，多次出现在类别定义中。

在受保护后声明的任何类成员：访问说明符 (直到下一个访问说明符)仅对该类的成员函数和类后代的成员函数可用。当尝试从外部引用具有private和protected说明符的成员时，我们得到编译阶段的错误。示例：

```
class A
  {
protected:
   //--- 复制操作符仅在类A及其后代中可用
   void operator=(const A &)
     {
     }
  };
class B
  {
   //--- 声明的类A对象
   A                 a;
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- 声明两个B类型变量
   B b1, b2;
   //--- 尝试将一个对象复制到另一个对象
   b2=b1;
  }
```

编译代码时，收到错误消息 —尝试调用远程复制操作符：

```
attempting to reference deleted function 'void B::operator=(const B&)'   trash3.mq5
```

下面的第二个字符串提供更详细的描述 — 类B中的复制操作符被显式删除，因为类A不可用的复制操作符被调用：

```
   function 'void B::operator=(const B&)' was implicitly deleted because it invokes ir
```

© 2000-2025, MetaQuotes Ltd.

基本分类的接入成员在派生类通过继承进行定义。

## 'delete'说明符

delete说明符标记不能使用的类成员函数。这意味着，如果程序显式或隐式引用这个函数，则已经在编译阶段收到错误。例如，这个说明符可以使您的父方法在子类中不可用。如果我们在父类私有区域声明该函数，也可以收到相同的结果（在private部分声明）。这里，使用delete可使代码在后代级别更具可读性和可管理性。

```
class A
  {
public:
                     A(void) {value=5;};
   double            GetValue(void) {return(value);}
private:
   double            value;
  };
class B: public A
  {
   double            GetValue(void)=delete;
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明A类型变量
   A a;
   Print("a.GetValue()=", a.GetValue());
//--- 尝试从B类型变量获取值
   B b;
   Print("b.GetValue()=", b.GetValue()); //编译器在此字符串显示一个错误
  }
```

编译器消息：

```
attempting to reference deleted function 'double B::GetValue()'
   function 'double B::GetValue()' was explicitly deleted here
```

'delete'说明符可以禁用自动转换或复制构造函数，否则它也必须隐藏在private部分中。例如：

```
class A
  {
public:
   void              SetValue(double v) {value=v;}
   //--- 禁用int类型调用
   void              SetValue(int) = delete;
   //--- 禁用复制操作符
   void              operator=(const A&) = delete;
private:
   double            value;
```

© 2000-2025, MetaQuotes Ltd.

```
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明两个A类型变量
   A a1, a2;
   a1.SetValue(3);       // 错误!
   a1.SetValue(3.14);    // OK
   a2=a1;                // 错误!
  }
```

编译期间，我们获得的错误消息：

```
attempting to reference deleted function 'void A::SetValue(int)'
   function 'void A::SetValue(int)' was explicitly deleted here
attempting to reference deleted function 'void A::operator=(const A&)'
   function 'void A::operator=(const A&)' was explicitly deleted here
```

### 修饰符 'final'

类声明过程中使用'final' 修饰符禁止从该类进一步继承。如果类界面无需进一步修改，或出于安全原因不允许修改，则以'final'修饰符声明该类。此外，该类的所有成员也将默认为不可更改。

```
class CFoo final
  {
  //--- Class body
  };

class CBar : public CFoo
  {
  //--- Class body
  };
```

如果您像上面示例一样试图以 'final' 修饰符继承形成一个类，编译器将返回一个错误：

```
不能像其被声明为'final'一样从'CFoo' 继承
参照 'CFoo' 声明
```

## 联合(union)

联合是一种特殊的数据类型，由多个共享相同内存区的变量组成。因此，联合提供了以两种 (或以上)不同方式解释相同位序列的能力。联合声明类似于结构声明，以关键词union开始。

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

与结构不同，不同的联合成员归属于相同的内存区。在这个示例中，LongDouble联合是通过long和double这两个类型值共享相同内存区来声明的。 请注意，联合不可以同时存储long整型值和

© 2000-2025, MetaQuotes Ltd.

*double*真实值（不同于结构体），因为long_value和double_value会部分重叠（内存中）。另一方面，MQL5程序能够随时以整型(long)或真实型(double)值处理包含在联合中的数据。因此，联合可以接收两种（或以上）选项来表示相同的数据序列。

联合声明期间，编译器会自动分配足够的内存区，以便在变量联合中存储最大的类型（根据交易量计算）。与结构同样的语法也用于访问联合元素 -点操作符。

```
union LongDouble
{
  long   long_value;
  double double_value;
};
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   LongDouble lb;
//--- 得到并显示无效 -nan(ind)数字
   lb.double_value=MathArcsin(2.0);
   printf("1.  double=%f                integer=%I64X",lb.double_value,lb.long_value);
//--- 最大的标准化值 (DBL_MAX)
   lb.long_value=0x7FEFFFFFFFFFFFFF;
   printf("2.  double=%.16e  integer=%I64X",lb.double_value,lb.long_value);
//--- 最小的标准化正值 (DBL_MIN)
   lb.long_value=0x0010000000000000;
   printf("3.  double=%.16e  integer=%.16I64X",lb.double_value,lb.long_value);
  }
/*  Execution result
    1.  double=-nan(ind)                integer=FFF8000000000000
    2.  double=1.7976931348623157e+308  integer=7FEFFFFFFFFFFFFF
    3.  double=2.2250738585072014e-308  integer=0010000000000000
*/
```

由于联合允许程序以不同的方式解释相同的内存数据，因此当需要不寻常的类型转换时通常会使用到它们。

联合不能参与到继承中，它们也不能有静态成员，因为这就是它们的本质。在所有其他方面，*联合*如同结构一样，其所有成员都是零点偏移。以下类型不能成为联合成员：

- 动态数组
- 字符串
- □□指针和函数
- 类对象
- 具有构造函数或析构函数的结构对象
- 具有来自1-5点成员的结构对象

© 2000-2025, MetaQuotes Ltd.

与类相似，联合既可以具有构造函数和析构函数，也可以拥有方法。默认情况下，联合成员具有public访问类型。若要创建私人元素，请使用private关键词。所有这些可能性都显示在这个示例中，说明了如何将color类型的颜色转换为ARGB，就像在ColorToARGB()函数一样。

```
//+------------------------------------------------------------------+
//| 颜色(BGR)联合转换为ARGB                                          |
//+------------------------------------------------------------------+
union ARGB
  {
   uchar             argb[4];
   color             clr;
   //--- 构造函数
                     ARGB(color col,uchar a=0){Color(col,a);};
                    ~ARGB(){};
   //--- 公共方法
public:
   uchar   Alpha(){return(argb[3]);};
   void    Alpha(const uchar alpha){argb[3]=alpha;};
   color   Color(){ return(color(clr));};
   //--- 私人方法
private:
   //+------------------------------------------------------------------+
   //| 设置alpha通道值和颜色                                            |
   //+------------------------------------------------------------------+
   void    Color(color col,uchar alpha)
     {
      //--- 设置颜色到 clr 成员
      clr=col;
      //--- 设置 Alpha 组件值 - 不透明度
      argb[3]=alpha;
      //--- 交换R和B组件的字节 红和蓝）
      uchar t=argb[0];argb[0]=argb[2];argb[2]=t;
     };
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 0x55 意味着 55/255=21.6 % (0% - 完全透明)
   uchar alpha=0x55;
//--- 颜色类型表示为 0x00BBGGRR
   color test_color=clrDarkOrange;
//--- 这里接受ARGB联合字节值
   uchar argb[];
   PrintFormat("0x%.8X - here is how the 'color' type look like for %s, BGR=(%s)",
               test_color,ColorToString(test_color,true),ColorToString(test_color));
//--- ARGB类型表示为 0x00RRGGBB，RR和BB组件互换
   ARGB argb_color(test_color);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 复制字节数组
   ArrayCopy(argb,argb_color.argb);
//--- 这里就是ARGB显示效果
   PrintFormat("0x%.8X - ARGB representation with the alpha channel=0x%.2x, ARGB=(%d,%
               argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- 增加不透明度
   argb_color.Alpha(alpha);
//--- 尝试定义ARGB为'color'类型
   Print("ARGB как color=(",argb_color.clr,")  alpha channel=",argb_color.Alpha());
//--- 复制字节数组
   ArrayCopy(argb,argb_color.argb);
//--- 这里就是ARGB显示效果
   PrintFormat("0x%.8X - ARGB representation with the alpha channel=0x%.2x, ARGB=(%d,%
               argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- 查看ColorToARGB()函数结果
   PrintFormat("0x%.8X - result of ColorToARGB(%s,0x%.2x)",ColorToARGB(test_color,alph
               ColorToString(test_color,true),alpha);
  }
/* Execution result
   0x00008CFF - here is how the color type looks for clrDarkOrange, BGR=(255,140,0)
   0x00FF8C00 - ARGB representation with the alpha channel=0x00, ARGB=(0,255,140,0)
   ARGB as color=(0,140,255)  alpha channel=85
   0x55FF8C00 - ARGB representation with the alpha channel=0x55, ARGB=(85,255,140,0)
   0x55FF8C00 - result of ColorToARGB(clrDarkOrange,0x55)
*/
```

## 界面

界面允许定义实施类的特定功能。实际上，界面是不能包含任何成员的类，并可能没有构造函数和/或析构函数。界面中声明的所有函数类都是纯虚拟的，甚至不需明确定义。

界面使用'interface'关键字定义。例如：

```
//--- 用于描述动物的基本界面
interface IAnimal
  {
//--- 界面函数类默认公开访问
   void Sound();  // 动物产生的声音
  };
//+------------------------------------------------------------------+
//|  CCat 类继承自 IAnimal 界面                                      |
//+------------------------------------------------------------------+
class CCat : public IAnimal
  {
public:
                     CCat() { Print("Cat was born"); }
                    ~CCat() { Print("Cat is dead");  }
   //--- 实现 IAnimal 界面的Sound 函数类
   void Sound(){ Print("meou"); }
```

© 2000-2025, MetaQuotes Ltd.

```
  };
//+------------------------------------------------------------------+
//|  CDog 类继承自 IAnimal 界面                                      |
//+------------------------------------------------------------------+
class CDog : public IAnimal
  {
public:
                     CDog() { Print("Dog was born"); }
                    ~CDog() { Print("Dog is dead");  }
   //--- 实现 IAnimal 界面的Sound 函数类
   void Sound(){ Print("guaf"); }
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- IAnimal 类型对象的指针数组
   IAnimal *animals[2];
//--- 创建IAnimal 子类并保存指针到一个数组
   animals[0]=new CCat;
   animals[1]=new CDog;
//--- 为每个子类调用基本 IAnimal 界面的Sound() 类函数
   for(int i=0;i<ArraySize(animals);++i)
      animals[i].Sound();
//--- 删除对象
   for(int i=0;i<ArraySize(animals);++i)
      delete animals[i];
//--- 执行结果
/*
   Cat was born
   Dog was born
   meou
   guaf
   Cat is dead
   Dog is dead
*/
  }
```

类似抽象类，界面对象不能再没有继承的情况下创建。界面只能从其他界面继承并可以使一个父类。界面始终公开可见。

界面不能在类或架构声明中进行声明，但界面指针可以保存在变量类型void *。一般来说，任何类的对象指针都可以保存在变量类型void *。若要将void * 指针转换到特殊类的对象指针，请使用dynamic_cast 操作符。如果无法转换，dynam ic_cast 操作的结果将是NULL。

另见

面向对象的程序设计

© 2000-2025, MetaQuotes Ltd.

# 动态数组目标

## 动态数组

最大的四次元数组也可被声明。当定义一个动态数组，(方括号里第一对未知值数组)，编译器自动编译上述结构变量 (动态数组变量)为正确的初始化提供代码。

当声明程序在能见度之上数组自动释放。

示例：

```
double matrix[][10][20]; // 三维动态数组
ArrayResize(matrix,5);   // 设置第一维大小
```

## 静态数组

当有效阵列维数被定义后，编译器会再分配必要的内存，这种数组叫做静态数组。然而，编译器会为动态数组目标分配额外内存，目标和再分配的静态缓冲器 (存储数组的内存部分)有关。

创建动态数组目标是需要经过静态数组确定参数和部分功能的。

Examples:

```
double stat_array[5]; // 1维静态数组
some_function(stat_array);
...
bool some_function(double& array[])
  {
   if(ArrayResize(array,100)<0) return(false);
   ...
   return(true);
  }
```

## 在架构中的数组

当静态数组被描述成架构中的一员，动态数组目标不能创建，在windows API中是为了确保数据结构的兼容性。

然而，静态数组都能被描述成结构成员，也能通过MQL5功能，在此情况下，当通过一个动态数组临时目标的常量时，与静态数组连接—架构会员就会产生。

另见

数组函数, 初始变量, 虚拟范围和时间变量, 创建和删除目标

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量

向量类型是MQL5中的一种特殊数据类型，它支持向量运算。向量是double类型的一维数组。它是线性代数的基本概念之一，应用于多个科学领域，包括物理学、几何学等。向量用于求解线性方程组、3D图形和其他应用领域。向量可以相加和相乘。向量之间的长度或距离可以通过范数获得。在编程中，向量通常用同构元素的数组表示，这些数组可能没有规则的向量运算，即数组不能相加或相乘时，它们没有范数。

使用矩阵时，向量可以表示为行向量和字符串向量。此外，线性代数中的向量使用协方差和逆变的概念。这些概念在编写MQL5代码时没有任何区别，因为只有程序员能决定每个向量类型的对象。例如，它可以是3D图形中的旋转、位移或压缩向量。

一般来说，从线性代数的角度来看，一个数也是一个向量，只不过是在一维向量空间中。向量本身可以被认为是矩阵的特例。

矩阵类型是MQL5中的另一种特殊数据类型，用于表示矩阵。矩阵实际上是double类型的二维数组。向量和矩阵已被引入MQL5，以便更轻松地对某些类型的数据集进行运算。通过向量和矩阵，开发人员可以以简单且类似于数学的形式从线性代数的可能性中受益。矩阵可用来更简洁地写出线性方程组或微分方程组。矩阵行数对应方程数，而列数等于未知数的个数。因此，线性方程组可以通过矩阵运算来求解。

为矩阵定义以下代数运算：

- 添加相同大小的矩阵
- 合适大小矩阵的乘法：左矩阵的列数必须等于右矩阵的行数
- 矩阵乘以列向量；根据矩阵乘法规则将行向量乘以矩阵。在这个意义上，向量是矩阵的特例
- 矩阵乘以一个数字，也就是标量

数学考虑了许多不同的矩阵类型。例如单位矩阵、对称矩阵、斜对称矩阵、上下三角矩阵等。各种规范形式在矩阵理论中扮演着重要的角色。它们代表矩阵的某种规范形式，可以通过某些变换来获得。在实践中，使用具有附加属性（例如稳定性）的规范形式。

使用向量和矩阵，或者更确切地说，使用相关类型的特殊方法，可以创建更简单、更简洁和更清晰的代码，这接近于数学符号。使用这些方法，您可以避免创建嵌套循环或在计算中注意正确索引数组。因此，使用这些方法可以提高开发复杂程序的可靠性和速度。

## 矩阵和向量方法列表

矩阵和向量类型包括对应于相关NumPy程序库方法的方法。使用这些方法，您可以轻松地将算法和代码从Python转换为MQL5。许多数据处理任务、数学方程、神经网络和机器学习任务都可以使用现有的Python方法和程序库来解决。

| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| void matrix.Eye(const int rows, const int cols, const int ndiag=0) | eye | 构造对角线为1，其他地方为0的矩阵 |
| void matrix.Identity(const int rows) | identity | 构造主对角线为1的矩形矩阵 |

© 2000-2025, MetaQuotes Ltd.

| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| void matrix.Ones(const int rows, const int cols) | ones | 构造给定行和列的新矩阵，用1填充 |
| void matrix.Zeros(const int rows, const int cols) | zeros | 构造给定行和列的新矩阵，用零填充 |
| void matrix.Full(const int rows, const int cols, const scalar value) | full | 构造给定行和列的新矩阵，用标值填充 |
| void matrix.Copy(const matrix a) | copy | 构造给定矩阵的副本 |
| void matrix.FromBuffer(const int rows, const int cols, const scalar array[], const int count=-1, const int offset=0) | frombuffer | 构造从一维数组创建的矩阵 |
| void matrix.FromFile(const int rows, condt int cols, const int file_handle, const int count=-1, const int offset=0) | fromfile | 构造从文本或二进制文件创建的矩阵 |
| void vector.FromString(const string source, const string sep=" ") | fromstring | 构造从字符串中的文本数据初始化的向量 |
| void vector.Arange(const scalar start, const scalar stop, const scalar step=1) | arange | 构造给定的间隔内均匀间隔的值 |
| void matrix.Diag(const vector v, const int ndiag=0) | diag | 提取对角线或构造对角矩阵 |
| void matrix.Tri(const int rows, const int cols, const int ndiag=0) | tri | 构造在给定对角线处和给定对角线以下处为1，其他地方为0的矩阵 |
| void matrix.Tril(const int rows, const int cols, const scalar array[], const int ndiag=0) | tril | 返回矩阵副本，其中第k条对角线以上的元素归零 |
| void matrix.Triu(const int rows, const int cols, const scalar array[], const int ndiag=0) | triu | 返回矩阵副本，其中第k对角线以下的元素归零 |
| void matrix.Vander(const vector v, const int cols=-1, const bool increasing=false) | vander | 生成Vanderm onde矩阵 |
| vector matrix.Row(const unsigned nrow) | | 返回行向量 |
| vector matrix.Col(const unsigned ncol) | | 返回列向量 |

© 2000-2025, MetaQuotes Ltd.

| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| unsigned matrix.Rows() | | 返回矩阵中的行数 |
| unsigned matrix.Cols() | | 返回矩阵中的列数 |
| void matrix.Init() | | 初始化矩阵 |
| matrix matrix.Transpose() | transpose | 反转或置换矩阵轴；返回修改后的矩阵 |
| matrix matrix.Dot(const matrix b) | dot | 两个矩阵的点积 |
| matrix matrix.Inner(const matrix b) | inner | 两个矩阵的内积 |
| matrix matrix.Outer(const matrix b) | outer | 计算两个矩阵的外积 |
| matrix matrix.MatMul(const matrix b) | matmul | 两个矩阵的矩阵乘积 |
| matrix matrix.MatrixPower(const int power) | matrix_power | 将矩形矩阵提升到（整数）n次方 |
| matrix matrix.Kron(const matrix b) | kron | 返回两个矩阵的Kronecker乘积 |
| bool matrix.Cholesky(matrix& L) | cholesky | 返回Cholesky分解 |
| bool matrix.QR(matrix& Q, matrix& R) | qr | 计算矩阵的qr因式分解 |
| bool matrix.SVD(matrix& U, matrix& V, vector& singular_values) | svd | 奇异值分解 |
| bool matrix.Eig(matrix& eigen_vectors, vector& eigen_values) | eig | 计算矩形矩阵的特征值和右特征向量 |
| bool matrix.EigH(matrix& eigen_vectors, vector& eigen_values) | eigh | 返回Hermitian矩阵的特征值和特征向量 |
| bool matrix.EigVals(vector& eigen_values) | eigvals | 计算一般矩阵的特征值 |
| bool matrix.EigValsH(vector& eigen_values) | eigvalsh | 计算Hermitian矩阵的特征值 |
| bool matrix.LU(matrix& L, matrix& U) | | 将矩阵LU分解为一个下三角形矩阵和上三角形矩阵的乘积 |
| bool matrix.LUP(matrix& L, matrix& U, matrix& P) | | 带部分枢轴的LUP分解，仅指带行排列的LU分解：PA=LU |
| double matrix.Norm(const norm) | norm | 返回矩阵或向量范数 |
| double matrix.Cond(const norm) | cond | 计算矩阵的条件数 |

© 2000-2025, MetaQuotes Ltd.

| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| vector matrix.Spectrum() | | 将矩阵的范围计算为来自AT*A乘积的特征值集合 |
| double matrix.Det() | det | 计算数组的行列式 |
| int matrix.Rank() | matrix_rank | 使用Gaussian方法返回数组的矩阵秩 |
| int matrix.SLogDet(int& sign) | slogdet | 计算数组行列式的符号和对数 |
| double matrix.Trace() | trace | 返回沿矩阵对角线的总和 |
| vector matrix.Solve(const vector b) | solve | 求解线性矩阵方程或线性代数方程组 |
| vector matrix.LstSq(const vector b) | lstsq | 返回线性代数方程组的最小平方解（对于非平方矩阵或退化矩阵） |
| matrix matrix.Inv() | inv | 计算矩阵的（乘法）逆矩阵 |
| matrix matrix.PInv() | pinv | 通过Moore-Penrose方法计算矩阵的伪逆 |
| int matrix.Compare(const matrix matrix_c, const double epsilon)<br>int matrix.Compare(const matrix matrix_c, const int digits)<br>int vector.Compare(const vector vector_c, const double epsilon)<br>int vector.Compare(const vector vector_c, const int digits) | | 以指定精度比较两个矩阵/向量的元素 |
| double matrix.Flat(const ulong index)<br>bool matrix.Flat(const ulong index,const double value) | flat | 允许通过一个索引而不是两个索引来寻址矩阵元素 |
| double vector.ArgMax()<br>double matrix.ArgMax()<br>vector matrix.ArgMax(const int axis) | argmax | 返回最大值的索引 |
| double vector.ArgMin()<br>double matrix.ArgMin()<br>vector matrix.ArgMin(const int axis) | argmin | 返回最小值的索引 |
| double vector.Max()<br>double matrix.Max()<br>vector matrix.Max(const int axis) | max | 返回矩阵/向量的最大值 |
| double vector.Mean()<br>double matrix.Mean()<br>vector matrix.Mean(const int axis) | 平均值 | 计算元素值的算术平均值 |

© 2000-2025, MetaQuotes Ltd.

| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| double vector.Min()<br>double matrix.Min()<br>vector matrix.Min(const int axis) | min | 返回矩阵/向量的最小值 |
| double vector.Sum()<br>double matrix.Sum()<br>vector matrix.Sum(const int axis) | sum | 返回可以对给定轴（坐标轴）执行的矩阵/向量元素的和。 |
| void vector.Clip(const double min_value,const double max_value)<br>void matrix.Clip(const double min_value,const double max_value) | clip | 将矩阵/向量的元素限制在指定的有效值范围内 |
| vector vector.CumProd()<br>vector matrix.CumProd()<br>matrix matrix.CumProd(const int axis) | cumprod | 返回矩阵/向量元素的累积乘积，包括沿给定轴的元素 |
| vector vector.CumSum()<br>vector matrix.CumSum()<br>matrix matrix.CumSum(const int axis) | cumsum | 返回矩阵/向量元素的累积和，包括沿给定轴的元素 |
| double vector.Prod(const double initial=1)<br>double matrix.Prod(const double initial=1)<br>vector matrix.Prod(const int axis,const double initial=1) | prod | 返回可以对给定轴执行的矩阵/向量元素的乘积。 |
| void matrix.Reshape(const ulong rows, const ulong cols) | reshape | 改变矩阵的形状而不改变其数据 |
| void matrix.Resize(const ulong rows,const ulong cols) | resize | 返回已改变形状和大小的新矩阵 |
| bool matrix.SwapRows(const ulong row1, const ulong row2) | | 交换矩阵中的行 |
| bool matrix.SwapCols(const ulong col1, const ulong col2) | | 交换矩阵中的列 |
| double vector.Ptp()<br>double matrix.Ptp()<br>vector matrix.Ptp(const int axis) | ptp | 返回矩阵/向量或给定矩阵轴的值范围，相当于Max()-Min() |
| double vector.Percentile(const int percent)<br>double matrix.Percentile(const int percent)<br>vector matrix.Percentile(const int percent,const int axis) | percentile | 返回矩阵/向量元素值或沿指定轴的元素值的指定百分位数。'percent'参数的有效值在[0, 100]范围内 |

© 2000-2025, MetaQuotes Ltd.

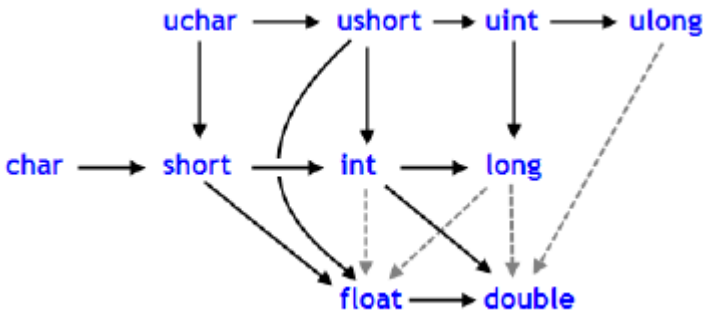
| 方法矩阵/向量 | NumPy中的类似方法 | 描述 |
| --- | --- | --- |
| double vector.Quantile(const int percent)<br>double matrix.Quantile(const int percent)<br>vector matrix.Quantile(const int percent,const int axis) | quantile | 返回矩阵/向量元素值或沿指定轴的元素值的指定分位数。'percent'参数的取值范围为[0, 1] |
| double vector.Median()<br>double matrix.Median()<br>vector matrix.Median(const int axis) | median | 计算矩阵/向量元素的中位数。中位数是将数组/向量元素的最高半部分与元素的最低半部分分开的中间值。 |
| double vector.Average()<br>double matrix.Average()<br>vector matrix.Average(const int axis) | average | 计算矩阵/向量值的算术平均值。分母中权重之和不能等于0，但有些权重可以为0 |
| double vector.Std()<br>double matrix.Std()<br>vector matrix.Std(const int axis) | std | 返回矩阵/向量元素值或沿给定轴的元素值的标准偏差 |
| double vector.Var()<br>double matrix.Var()<br>vector matrix.Var(const int axis) | var | 计算矩阵/向量元素值的方差 |
| double vector.CorrCoef(const vector& v)<br>matrix matrix.CorrCoef() | corrcoef | 计算Pearson相关系数（线性相关系数）。相关系数在[-1, 1]范围内 |
| vector vector.Correlate(const vector& v,enum mode) | correlate | 计算两个向量的交叉相关。'mode'参数决定线性卷积计算模式 |
| vector vector.Convolve(const vector& v, enum mode) | convolve | 返回两个向量的离散、线性卷积'mode'参数决定线性卷积计算模式 |
| matrix matrix.Cov()<br>matrix vector.Cov(const vector& v); (resulting matrix 2 x 2) | cov | 计算协方差矩阵。两个样本（两个随机变量）的协方差是衡量它们线性相关性的指标 |

© 2000-2025, MetaQuotes Ltd.

# 类型转换

## 创建数字符

有必要把一组数字类型变化成另一种数字类型，但并非作用数字类型都能转换，下面是允许转换的模式：




箭头指明表示转换方向，期间没有任何损失信息，布尔型可以取代字符类型（只占用1字节），颜色型可以取代整型（4字节），日期时间型可以取代长型（占用8字节）。四条灰色虚线，也带有箭头，在精确度缺失时，表示转化。例如，与123456789相等的整数字（int）就高于浮点型表示的数字。

```
   int n=123456789;
   float f=n;     //  f 内容同于 1.234567892E8
   Print("n = ",n,"    f = ",f);
   // 结果 n= 123456789     f= 123456792.00000
```

转化成浮点型的数字有同样的命令，但是精确度比较低。与黑色箭头不同的是，转化允许部分数据丢失。字符型和无符号字符型间的转化，短整型和无符号短整型间的转化，整型及无符号整型的转化，长整型和无符号长整型的转化（双向转化），都可能导致数据丢失。

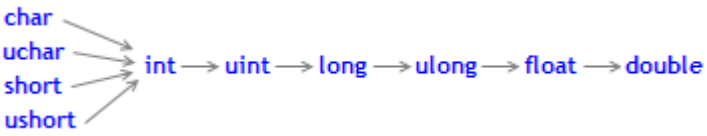
因此浮点值转化成整数型的结果就是，经常删除小数部分。如果想把浮点转成最近的整数（任何情况下都很有用），应该使用 MathRound() 。

示例：

```
//--- 重力加速度
   double g=9.8;
   double round_g=(int)g;
   double math_round_g=MathRound(g);
   Print("round_g = ",round_g);
   Print("math_round_g = ",math_round_g);
/*
   Result:
   round_g = 9
   math_round_g = 10
*/
```

如果用二进制合并两个值，执行操作前，需要按照下表的先后顺序，把低类型转化成高类型。

© 2000-2025, MetaQuotes Ltd.




数字类型字符型，无符号字符型，短整型，和无符号短整型，无条件的转化成整型。

示例：

```
   char    c1=3;
//--- 第一示例
   double d2=c1/2+0.3;
   Print("c1/2 + 0.3 = ",d2);
// 结果:    c1/2+0.3 = 1.3

//--- 第二示例
   d2=c1/2.0+0.3;
   Print("c1/2.0 + 0.3 = ",d2);
// 结果:    c1/2.0+0.3 = 1.8
```

计算的表达式由两种操作构成。示例一，字符型变量c1转化成整型临时变量，因为除法操作中的第二运算对象，常量2，是高类型整型。因此3/2的整数我们取整数值，1。

在示例一中的第二步中，第二运算对象是常量0.3，双精度型，那么结果就是第一运算对象转化成1.0双精度型临时变量。

示例二中，字符型c1变量转化成双精度型临时变量，因为除法操作的第二运算对象，常量2.0，是双精度型；无进一步转化。

## 数型类型转换

在MQL5语言表达式中，使用明确和含蓄两种类型转换。明确类型转换如下：

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

表达式或者函数执行的结果可用作 var_2变量。明确类型转换的函数记录也可以如此：

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

基于第一示例，考虑下明确类型转换。

```
//--- 第三示例
   double d2=(double)c1/2+0.3;
   Print("(double)c1/2 + 0.3 = ",d2);
// 结果:    (双精度)c1/2+0.3 = 1.80000000
```

做除法前，c1变量明确为双精度型。现在整型常量2，转换成双精度型2.0，因为转换造成第一运算对象成为双精度型。实际上，明确类型转换时是一种一元运算操作。

此外，当尝试转换类型时，结果可能超出允许范围内。在这个情况下，容易发生截断。例如：

```
   char c;
```

© 2000-2025, MetaQuotes Ltd.

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

在运算完成之前（除了数据已被定义的），数据会根据优先级被转换。当定义数据的操作完成前，数据会转换成被定义的数据类型。

示例：

```
   int     i=1/2;          // 无类型转换, 结果是 0
   Print("i = 1/2  ",i);

   int k=1/2.0;            // 表达式转换到双精度型,
   Print("k = 1/2  ",k);   // 那么就是到整型的目标类型,结果是0

   double d=1.0/2.0;       // 无类型转换,结果是 0.5
   Print("d = 1/2.0; ",d);

   double e=1/2.0;         // 表达式转换到双精度型,
   Print("e = 1/2.0; ",e);// 同于目标类型,结果为0.5

   double x=1/2;           // 整型表达式转换到双精度目标类型,
   Print("x = 1/2; ",x);   // 结果是 0.0
```

如果整型值大于9223372036854774784或小于-9223372036854774784，当从长整型/无符号长整型转化到双精度型时，精度可能会丢失。

```
void OnStart()
  {
   long l_max=LONG_MAX;
   long l_min=LONG_MIN+1;
//--- 定义最高整型值,在转换到双精度时不会丢失精度。
   while(l_max!=long((double)l_max))
      l_max--;
//--- 定义最低整型值,在转换到双精度时不会丢失精度。
   while(l_min!=long((double)l_min))
      l_min++;
//--- 派生发现的整型值区间
   PrintFormat("When casting an integer value to double, it must be "
               "within [%I64d, %I64d] interval",l_min,l_max);
//--- 现在,让我们看看如果值跌落该区间会发生什么
   PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))=%I64d",
               l_max+1,double(l_max+1),long(double(l_max+1)));
   PrintFormat("l_min-1=%I64d, double(l_min-1)=%.f, ulong(double(l_min-1))=%I64d",
               l_min-1,double(l_min-1),long(double(l_min-1)));
//--- 收到下面结果
// 当将整型值转换到双精度型时,它应该在[-9223372036854774784, 9223372036854774784]区间。
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong(double(l_ma
```

© 2000-2025, MetaQuotes Ltd.

```
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulong(double(l_
  }
```

## 字符串类型转换

字符串类型是几种简单类型中的最高级别。因此，如果操作的运算对象之一为字符串，第二运算对象自动转换成字符串。注意的是，对于字符串，添加独立二元操作是可以的。允许任何字符串明确转换成数字类型。

示例：

```
   string s1=1.0/8;              // 表达式转换到双精度型,
   Print("s1 = 1.0/8; ",s1);      //  那么就是到字符串的目标类型,
// 结果 is "0.12500000" (包括10个字符的字符串)

   string s2=NULL;               // 字符串无法初始化
   Print("s2 = NULL; ",s2);       // 结果是空值字符串
   string s3="Ticket N"+12345; // 表达式转换到字符串类型
   Print("s3 = \"Ticket N\"+12345",s3);

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

## 基本类指针到派生类指针的类型转换

打开生成分类目标也可以看做相关基本类目标。这将导致一些有趣的影响。例如，即使一个基本类生成的不同类目标彼此明显不同，我们仍然可以创建它们的链接列表(List)，因为我们将它们全部视为基本类型的目标。但是反过来却不可以：基本类目标不能自动成为派生类的目标。

您可以使用明确转换，将基本类指针转化成派生类指针。但是对这种转化要有足够的资格，因为否则的话，会导致危险的运行错误而mq5程序会停止。

## 动态类型转换使用 dynamic_cast 操作符

动态类型转换使用仅能用于指针到类的dynamic_cast操作符来执行。在运行时完成类型验证。这意味着使用dynamic_cast操作符时编译器不会检查应用于类型转换的数据类型。如果指针转换到一个并不是实际对象类型的数据类型，那么结果为NULL。

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

© 2000-2025, MetaQuotes Ltd.

尖括号中的 *type-id* 参数应该指向之前定义的类类型。不同于 C++，*表达式* 操作数类型可以是除了void以外的任何值。

例如：

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

void OnStart()
  {
   CBar bar;
//--- 允许动态转换* bar 指针类型 * foo 指针
   CFoo *foo = dynamic_cast<CFoo *>(&bar); // 不重要的错误
   Print(foo);                             // foo=NULL
//--- 禁止尝试引用Foo 类型对象明确转换Bar 类型对象
   foo=(CFoo *)&bar;                       // 关键的运行时间错误
   Print(foo);                             // 这个字符串不被执行
  }
```

另见

数据类型

© 2000-2025, MetaQuotes Ltd.

# 空型和NULL常量

依照语法来说，空型是一种基本类型，附带图表、非图表、布尔型、短整型、无符号短整型、整型、无符号整型、长整型、无符号长整型、日期时间型、浮点型、双精度型和字符串型。类型被指示出来，该功能不返回任何值，而一个功能参数表示一个出席参数。

预先参数变量**NULL**是空型值，能无需转换标识任何其他基本类型的常量，允许比较基本类型变量与**NULL**值。

示例：

```
//--- 如果字符串没有初始化，那么会分配预先定义的值给它
if(some_string==NULL) some_string="empty";
```

**NULL**也能与指示目标中的 新操作 对比。

另见

变量，函数

© 2000-2025, MetaQuotes Ltd.

# 用户定义类型

在C++中，关键词typedef允许创建用户定义数据类型。因此，只需为现有的数据类型指定一个新的数据类型名称即可。不可以创建新的数据类型。而是用现有类型的新名称来替代。用户定义类型可以使应用程序更具灵活性：有时，使用替换宏(#define)就足以改变 typedef 指令。用户定义类型还可以提高代码的可读性，因为可以使用typedef将自定义名称应用于标准数据类型。创建用户类型条目的一般格式：

```
   typedef type new_name;
```

在这里，*type* 表示任何可接受的数据类型，而*new_name*是该类型的新名称。设置的新名称只能作为现有类型名称的补充（而不能替换它）。MQL5 允许使用typedef创建函数指针。

## 函数指针

函数指针通常根据以下格式定义

```
   typedef function_result_type (*Function_name_type)(list_of_input_parameters_types);
```

typedef之后，设置函数签名（输入参数的数字和类型，以及函数返回结果的类型）。下面就是创建和应用函数指针的简单示例：

```
//--- 声明接受两种int参数的函数指针
   typedef int (*TFunc)(int,int);
//--- TFunc 是一种类型，可以声明变量函数指针
   TFunc func_ptr; // pointer to the function
//--- 声明符合TFunc 描述的函数
   int sub(int x,int y) { return(x-y); }  // 从一个数减去一个数
   int add(int x,int y) { return(x+y); }  // 两个数字相加
   int neg(int x)       { return(~x);  }  // 逆位变量
//--- func_ptr 变量可以存储函数地址以便稍后声明
   func_ptr=sub;
   Print(func_ptr(10,5));
   func_ptr=add;
   Print(func_ptr(10,5));
   func_ptr=neg;            // 错误:neg 没有 int (int,int) 类型
   Print(func_ptr(10));     // 错误:需要两个参数
```

在该示例中，*func_ptr*变量可以接收*sub*和*add*函数，因为它们具有如*TFunc*函数指针定义的每个int类型的两个输入参数。相反，因为签名不同，所以*neg*函数不可以安排到func_ptr指针。

### 在用户界面安排事件模式

函数指针允许您创建用户界面时轻松创建事件处理。让我们通过CButton版块的示例来说明如何创建按键和添加函数来处理它们。首先，定义一个通过按下按键来调用的*TAction*函数并根据*TAction描述创建三个函数*。

```
//--- 创建一个自定义函数类型
typedef int(*TAction)(string,int);
//+------------------------------------------------------------------+
//|  打开文件                                                        |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int Open(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(1);
  }
//+------------------------------------------------------------------+
//|  保存文件                                                        |
//+------------------------------------------------------------------+
int Save(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(2);
  }
//+------------------------------------------------------------------+
//|  关闭文件                                                        |
//+------------------------------------------------------------------+
int Close(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(3);
  }
```

然后，创建来自CButton的MyButton类，在这里我们应该添加*TAction*函数指针。

```
//+------------------------------------------------------------------+
//| 通过事件处理函数创建按键类                                       |
//+------------------------------------------------------------------+
class MyButton: public CButton
  {
private:
   TAction           m_action;                    // 图表事件处理程序
public:
                     MyButton(void){}
                    ~MyButton(void){}
   //--- 构造函数指定按键文本和事件处理函数指针
                     MyButton(string text, TAction act)
     {
      Text(text);
      m_action=act;
     }
   //--- 设置OnEvent()事件处理程序调用的自定义函数
   void              SetAction(TAction act){m_action=act;}
   //--- 标准图表事件处理程序
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
     {
      if(m_action!=NULL && lparam==Id())
        {
         //--- 调用自定义 m_action() 处理程序
```

© 2000-2025, MetaQuotes Ltd.

```
         m_action(sparam,(int)lparam);
         return(true);
        }
      else
      //--- 返回调用CButton父类处理程序的结果
         return(CButton::OnEvent(id,lparam,dparam,sparam));
     }
  };
```

创建来自CAppDialog的CControlsDialog衍生类，为其添加*m_buttons*数组来存储*MyButton*类型的按键，以及 *AddButton(MyButton &button)* 和 *CreateButtons()* 方法。

```
//+------------------------------------------------------------------+
//| CControlsDialog 类                                               |
//| 目标:管理应用程序的图形面板                                      |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CArrayObj         m_buttons;                    // 按键数组
public:
                     CControlsDialog(void){};
                    ~CControlsDialog(void){};
   //--- 创建
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- 添加按键
   bool              AddButton(MyButton &button){return(m_buttons.Add(GetPointer(butto
protected:
   //--- 创建按键
   bool              CreateButtons(void);
  };
//+------------------------------------------------------------------+
//| 在图表上创建CControlsDialog 对象                                 |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
   return(CreateButtons());
//---
  }
//+------------------------------------------------------------------+
//| 定义                                                             |
//+------------------------------------------------------------------+
//--- 缩进和间隔
#define INDENT_LEFT                         (11)      // 从左缩进 预留边框宽度）
#define INDENT_TOP                          (11)      // 从上缩进 预留边框宽度）
#define CONTROLS_GAP_X                      (5)       // X坐标间隔
#define CONTROLS_GAP_Y                      (5)       // Y坐标间隔
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 按键
#define BUTTON_WIDTH                    (100)      // X坐标大小
#define BUTTON_HEIGHT                   (20)       // Y坐标大小
//--- 显示区域
#define EDIT_HEIGHT                     (20)       // Y坐标大小
//+------------------------------------------------------------------+
//| 创建和添加按键到CControlsDialog面板                                |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButtons(void)
  {
//--- 计算按键坐标
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2;
   int y2=y1+BUTTON_HEIGHT;
//--- 添加带有函数指针的按键对象
   AddButton(new MyButton("Open",Open));
   AddButton(new MyButton("Save",::Save));
   AddButton(new MyButton("Close",Close));
//--- 创建图形按键
   for(int i=0;i<m_buttons.Total();i++)
     {
      MyButton *b=(MyButton*)m_buttons.At(i);
      x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
      x2=x1+BUTTON_WIDTH;
      if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
         PrintFormat("Failed to create button %s %d",b.Text(),i);
         return(false);
        }
      //--- 添加每个按键到CControlsDialog 集合
      if(!Add(b))
         return(false);
     }
//--- 成功
   return(true);
  }
```

现在，我们可以使用包括3个按键（打开，保存和关闭）的CControlsDialog控制面板开发程序。当点击按键时，以*TAction*指针形式调用合适的函数。

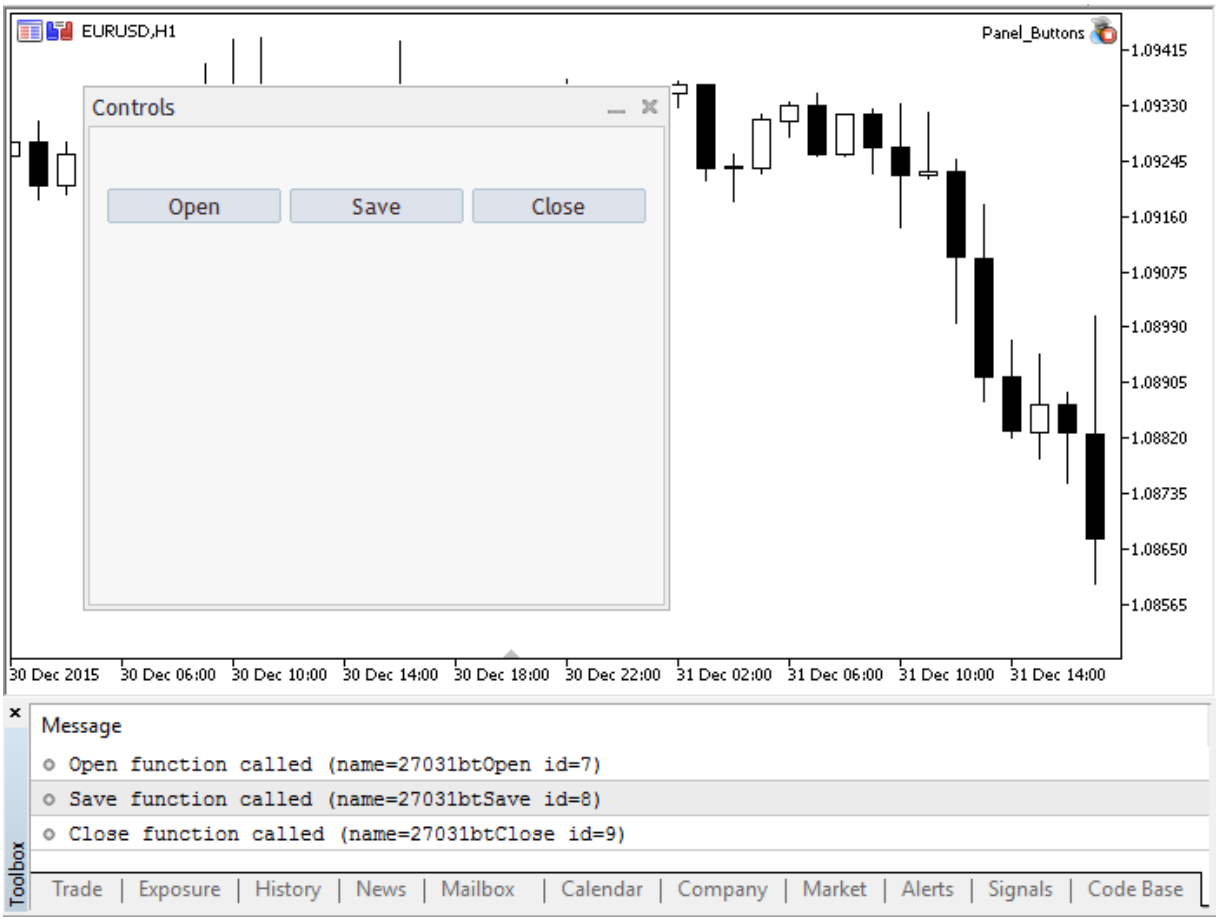
```
//--- 声明全局对象以便在启动程序时自动创建
CControlsDialog MyDialog;
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 现在，在图表上创建对象
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- 启动应用程序
   MyDialog.Run();
//--- 应用程序成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 销毁对话框
   MyDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| EA交易图表事件函数                                               |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // 事件 ID
                  const long& lparam,    // long型事件参数
                  const double& dparam, // double型事件参数
                  const string& sparam) // string型事件参数
  {
//--- 调用图表事件父类 这里是CAppDialog)的处理程序
   MyDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

启动应用程序的外观和点击结果的按键在截图中提供。

© 2000-2025, MetaQuotes Ltd.




程序的完整源代码

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

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The panel with several CButton buttons"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+------------------------------------------------------------------+
//| 定义                                                             |
//+------------------------------------------------------------------+
//--- 缩进和间隔
#define INDENT_LEFT                         (11)      // 从左缩进 预留边框宽度）
#define INDENT_TOP                          (11)      // 从上缩进 预留边框宽度）
#define CONTROLS_GAP_X                      (5)       // X坐标间隔
#define CONTROLS_GAP_Y                      (5)       // Y坐标间隔
```

```
//--- 按键
#define BUTTON_WIDTH                    (100)      // X坐标大小
#define BUTTON_HEIGHT                   (20)       // Y坐标大小
//--- 显示区域
#define EDIT_HEIGHT                     (20)       // Y坐标大小

//--- 创建自定义函数类型
typedef int(*TAction)(string,int);
//+------------------------------------------------------------------+
//|  打开文件                                                        |
//+------------------------------------------------------------------+
int Open(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(1);
  }
//+------------------------------------------------------------------+
//|  保存文件                                                        |
//+------------------------------------------------------------------+
int Save(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(2);
  }
//+------------------------------------------------------------------+
//|  关闭文件                                                        |
//+------------------------------------------------------------------+
int Close(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(3);
  }
//+------------------------------------------------------------------+
//| 通过事件处理函数创建按键类                                       |
//+------------------------------------------------------------------+
class MyButton: public CButton
  {
private:
   TAction           m_action;                    // 图表事件处理程序
public:
                     MyButton(void){}
                    ~MyButton(void){}
   //--- 构造函数指定按键文本和事件处理函数指针
                     MyButton(string text,TAction act)
     {
      Text(text);
      m_action=act;
     }
   //--- 设置OnEvent()事件处理程序调用的自定义函数
```

© 2000-2025, MetaQuotes Ltd.

```
   void              SetAction(TAction act){m_action=act;}
   //--- 标准图表事件处理程序
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
     {
      if(m_action!=NULL && lparam==Id())
        {
         //--- 调用自定义处理程序
         m_action(sparam,(int)lparam);
         return(true);
        }
      else
      //--- 返回调用CButton父类处理程序的结果
         return(CButton::OnEvent(id,lparam,dparam,sparam));
     }
  };
//+------------------------------------------------------------------+
//| CControlsDialog 类                                               |
//| 目标:管理应用程序的图形面板                                      |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CArrayObj         m_buttons;                    // 按键数组
public:
                     CControlsDialog(void){};
                    ~CControlsDialog(void){};
   //--- 创建
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- 添加按键
   bool              AddButton(MyButton &button){return(m_buttons.Add(GetPointer(butto
protected:
   //--- 创建按键
   bool              CreateButtons(void);
  };
//+------------------------------------------------------------------+
//| 在图表上创建CControlsDialog 对象                                 |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
   return(CreateButtons());
//---
  }
//+------------------------------------------------------------------+
//| 创建和添加按键到CControlsDialog面板                              |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButtons(void)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 计算按键坐标
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2;
   int y2=y1+BUTTON_HEIGHT;
//--- 添加带有函数指针的按键对象
   AddButton(new MyButton("Open",Open));
   AddButton(new MyButton("Save",::Save));
   AddButton(new MyButton("Close",Close));
//--- 创建图形按键
   for(int i=0;i<m_buttons.Total();i++)
     {
      MyButton *b=(MyButton*)m_buttons.At(i);
      x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
      x2=x1+BUTTON_WIDTH;
      if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
         PrintFormat("Failed to create button %s %d",b.Text(),i);
         return(false);
        }
      //--- 添加每个按键到CControlsDialog 集合
      if(!Add(b))
         return(false);
     }
//--- 成功
   return(true);
  }
//--- 声明全局对象以便在启动程序时自动创建
CControlsDialog MyDialog;
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 现在，在图表上创建对象
   if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- 启动应用程序
   MyDialog.Run();
//--- 应用程序成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 销毁对话框
   MyDialog.Destroy(reason);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| EA交易图表事件函数                                                 |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // 事件 ID
                  const long& lparam,    // long型事件参数
                  const double& dparam, // double型事件参数
                  const string& sparam) // string型事件参数
  {
//--- 调用图表事件父类 这里是CAppDialog)的处理程序
   MyDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

另 见

变量, 函数

© 2000-2025, MetaQuotes Ltd.

# 目标指针

MQL5支持动态创建复杂类型对象。这是通过使用'new'操作符来完成，该操作符返回已创建对象的描述符。描述符大小为8个字节。在语法上，MQL5中的对象描述符类似于C++指针。

例如：

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

与C++不同，上例中的'hobject'变量不是指向内存的指针，而是对象描述符指针。此外，在MQL5中，函数参数中的所有对象都必须通过引用传递。下面的示例展示了对象作为函数参数的传递：

```
class Foo
  {
public:
   string            m_name;
   int               m_id;
   static int        s_counter;
   //--- 构造函数和析构函数
                     Foo(void){Setup("noname");};
                     Foo(string name){Setup(name);};
                    ~Foo(void){};
   //--- 初始化Foo对象
   void              Setup(string name)
     {
      m_name=name;
      s_counter++;
      m_id=s_counter;
     }
  };
int Foo::s_counter=0;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 将对象声明为一个变量，并自动创建
   Foo foo1;
//--- 通过引用传递对象的变体
   PrintObject(foo1);

//--- 声明一个指向对象的指针，并使用'new'操作符进行创建
   Foo *foo2=new Foo("foo2");
//--- 通过引用传递指对象指针的变体
   PrintObject(foo2); // 对象指针由编译器自动转换

//--- 声明Foo对象的数组
   Foo foo_objects[5];
//--- 传递对象数组的变体
```

© 2000-2025, MetaQuotes Ltd.

```
   PrintObjectsArray(foo_objects); // 用于传递对象数组的单独函数

//--- 声明指向Foo类型对象的指针数组
   Foo *foo_pointers[5];
   for(int i=0;i<5;i++)
      foo_pointers[i]=new Foo("foo_pointer");
//--- 传递指针数组的变体
   PrintPointersArray(foo_pointers); // 用于传递指针数组的单独函数

//--- 完成之前，一定要删除作为指针创建的对象
   delete(foo2);
//--- 移除指针数组
   int size=ArraySize(foo_pointers);
   for(int i=0;i<5;i++)
      delete(foo_pointers[i]);
//---
  }
//+------------------------------------------------------------------+
//|  对象总是通过引用传递                                            |
//+------------------------------------------------------------------+
void PrintObject(Foo &object)
  {
   Print(__FUNCTION__,": ",object.m_id," Object name=",object.m_name);
  }
//+------------------------------------------------------------------+
//| 传递对象数组                                                     |
//+------------------------------------------------------------------+
void PrintObjectsArray(Foo &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
      PrintObject(objects[i]);
  }
//+------------------------------------------------------------------+
//| 传递对象指针数组                                                 |
//+------------------------------------------------------------------+
void PrintPointersArray(Foo* &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
      PrintObject(objects[i]);
  }
//+------------------------------------------------------------------+
```

## 使用前检查指针

© 2000-2025, MetaQuotes Ltd.

尝试访问无效指针会导致程序严重关闭。CheckPointer函数用于在使用前检查指针。指针在以下情况下可能无效：

- 指针等于NULL；
- 对象已使用delete操作符销毁。

此函数可用于验证指针。非零值表示可以在此指针处访问数据。

```
class CMyObject
 {
protected:
  double             m_value;
public:
                     CMyObject(void);
                     CMyObject(double value) {m_value=value;};
                    ~CMyObject(void){};
  //---
  double             Value(void) {return(m_value);}
 };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- 创建未初始化对象
  CMyObject *pointer;
  if(CheckPointer(pointer)==POINTER_INVALID)
    Print("1. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("1. pointer.Value()=", pointer.Value());

//--- 初始化指针
  pointer=new CMyObject(M_PI);
  if(CheckPointer(pointer)==POINTER_INVALID)
    Print("2. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("2. pointer.Value()=", pointer.Value());

//--- 删除对象
  delete(pointer);
  if(CheckPointer(pointer)==POINTER_INVALID)
    Print("3. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("3. pointer.Value()=", pointer.Value());
 }
/*
   1. pointer is POINTER_INVALID
   2. pointer.Value()=3.141592653589793
   3. pointer is POINTER_INVALID
*/
```

© 2000-2025, MetaQuotes Ltd.

要快速验证指针，您还可以使用操作符"!" (LNOT)通过隐式调用CheckPointer函数来检查。这允许更简洁和清晰的代码编写。以下是上一个示例中的检查：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- 创建未初始化对象
  CMyObject *pointer;
  if(!pointer)
    Print("1. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("1. pointer.Value()=", pointer.Value());

//--- 初始化指针
  pointer=new CMyObject(M_PI);
  if(!pointer)
    Print("2. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("2. pointer.Value()=", pointer.Value());

//--- 删除对象
  delete(pointer);
  if(!pointer)
    Print("3. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("3. pointer.Value()=", pointer.Value());
 }
/*
   1. pointer is POINTER_INVALID
   2. pointer.Value()=3.141592653589793
   3. pointer is POINTER_INVALID
*/
```

Operator "==" is used for a quick check for NULL. 例如：ptr==NULL或ptr!=NULL。

另见

变量，初始变量，变量可见范围和使用期限，创建和删除目标

© 2000-2025, MetaQuotes Ltd.

# 引用，修饰符& 和关键字 this

## 通过引用传送参数

在MQL5的简单类型参数能够通过值和引用传送，而复合类型参数经常通过引用传送。若使编译器了解参数是通过引用传送，需要在参数前加上 & 符号。

通过引用传送参数意味着传送变量地址，这就是通过引用传送的参数的所有改变能够马上在变量源中反映出来的原因。使用通过引用传送的参数，一个函数可以同时得出几种结果。为防止引用传送的参数改变，使用常量修饰符。

因此，函数输入参数是一个数组，结构或者类，符号'&'放在函数表头，变量类型后，名字前。

示例：

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

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

上面示例中，包括私人成员的类CDemoClass被声明-双精度型数组 m_array[] 。函数 setArray()被声明，通过引用传送array[]到这里。如果函数表头不包括通过引用传送的指示，例如，不包括& 符号，并且编译这个符号时会生成错误信息。

即使数组是通过引用传送的，我们仍不能把一个数组定义到另一个。我们需要将数组源里逐元素复制成可接受的数组。函数描述中& 是数组和结构传送成函数参数必要的状态。

## 关键字 this

类型 (object)变量可以通过引用和指针传送。如引用一样，指针允许访问目标。目标指针声明后，就会应用新操作去创建和初始化。

关键字**this**意在获得目标的引用，可以在类或者结构函数里得到。**this**经常引用到目标中，并在此函数中使用，表达式 GetPointer(this) 给出目标指针，其成员为完成GetPointer()的函数。在MQL5中，函数不能返回对象，但是它们能返回目标指针。

因此，如果我们需要函数返回对象，我们能够在GetPointer(this)形式下返回目标指针。我们将这个类的目标指针函数getDemoClass()添加到CDemoClass描述。

© 2000-2025, MetaQuotes Ltd.

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

public:
   void              setArray(double &array[]);
   CDemoClass       *getDemoClass();
  };
//+------------------------------------------------------------------+
//| 填充数组                                                         |
//+------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
      ArrayResize(m_array,ArraySize(array));
      ArrayCopy(m_array,array);
     }
  }
//+------------------------------------------------------------------+
//| 返回它自己的指针                                                 |
//+------------------------------------------------------------------+
CDemoClass *CDemoClass::getDemoClass(void)
  {
   return(GetPointer(this));
  }
```

结构没有指针，*新建和删除*操作不能应用其上，不可以用GetPointer(this)。

另见

目标指针，创建和删除对象，可见范围和变量使用期

© 2000-2025, MetaQuotes Ltd.

# 运行式和表达式

一些数字和字符的组合是特别重要的，它们被称为运算符，例如：

```
+ - * / %       算术运算符
&& ||           逻辑运算符
= += *=         负值运算符
```

运算符应用在表达式中实现特定的作用。需要特别注意标点符号如圆括号、方括号、逗号、冒号、分号。

运算符，标点符号，空格用于分割语句的不同部分。

这部分包括以下主题：

- 表达式
- 算术运算
- 指定运算
- 关系运算
- 布尔运算
- 逐位运算
- 其他运算
- 优先运算命令

© 2000-2025, MetaQuotes Ltd.

# 表达式

一个表达式可以拥有多个字符和操作符。一个表达式可以写在几行里面。

示例：

```
a++; b = 10;          // 几个表达式位于一行
//---  一个表达式被分成几行
x = (y * z) /
    (w + 2) + 127;
```

一个表达式的最后一个分号 (;)操作符。

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 算术运算

算术运算符包括加法和乘法运算：

```
变量总和                    i = j + 2;
变量差                      i = j - 3;
改变符号                    x = - x;
变量产品                    z = 3 * x;
除法系数                    i = j / 5;
除法余数                    minutes = time % 60;
添加1 到变量值              i++;
添加 1 到变量值             ++i;
从变量值减 1                k--;
从变量值减 1                --k;
```

加减运算只用于变量中，不能用于常量里。加减前缀 (++i) 和 (—k)只在变量用在表达式前才用于这个变量。

加减后缀 (i++)和(k—)只在变量用在表达式后才用于这个变量。

重要提示

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

当移动上面表达式从一个编程环境到另一个 (例如，从Borland C++到MQL5)时，计算问题可能会发生。通常，计算顺序取决于编译器的实施。实际情况下，有两种方式实施递减 (递增)：

1. 计算整个表达式后递减 (递增)应用于变量。
2. 操作时递减 (递增)立即应用于变量。

目前MQL5实施第一种递减 (递增)计算方式。但即使知道这个特点，也不建议尝试使用。

示例：

```
int a=3;
a++;             // 有效表达式
int b=(a++)*3;   // 无效表达式
```

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 赋值运算

包括给出运算式的表达式值就是赋值后留下的运算对象：

| | |
|---|---|
| 赋值 x 值到 y 变量 | y = x; |

下面的运算式将赋值运算式与算法和逐位运算相结合：

| | |
|---|---|
| 添加 x 到 y 变量 | y += x; |
| 从 y 变量减去 x | y -= x; |
| y 变量乘以 x | y *= x; |
| y 变量除以 x | y /= x; |
| y 变量除以x 后的 余数 | y %= x; |
| y 二进制表示法向右转换 x 比特 | y >>= x; |
| y 二进制表示法向左转换 x 比特 | y <<= x; |
| y 和 x 二进制表示法的AND 逐位运算 | y &= x; |
| y 和 x 二进制表示法的 OR 逐位运算 | y \|= x; |
| y 和 x 二进制表示法的 OR 除外的逐位运算 | y ^= x; |

逐位算法只能用于整数。当y与右/左x完成逻辑移位时，x使用最小的二进制值5，放弃最大值，例如0-31间的移位。

By %= 运算式 (x与y组件)，结果等于分开数字符。

赋值操作符在一个表达式中可以使用多次。这种情况下，表达式从左到右执行：

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

首先，x变量赋值3，y变量赋予x值，也是3。

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 关系运算

布尔运算 FALSE代表整数零值，而布尔运算TRUE代表不同于零的任何值。

用返回FALSE (0) 或者TRUE (1)来表示逻辑值 两个量之间的关系。

| | |
|---|---|
| 等于b | a == b; |
| 不等于b | a != b; |
| 小于b | a < b; |
| 大于b | a > b; |
| 小于等于b | a <= b; |
| 大于等于b | a >= b; |

相等的两个真实数字不能比较。大部分情况下，两个看起来相似的数字不能相同，因为15小数位的值不同。为了正确比较两个真实数字，用零比较这些不同。

示例：

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

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 布尔运算

## 否定运算符 (!)

否定运算符(!)，用来表示真假的反面的结果。如果运算值是 FALSE (0)结果为TRUE (1)；如果运算不同于FALSE (0)等于FALSE (0)。

```
if(!a) Print("不是 'a'");
```

## 逻辑运算符或 OR (||)

x和y值的逻辑运算符或OR(||)用来表示两个表达式只要有一个成立即可。如果x和y值为真的，表达式值为TRUE (1)。否则，值为FALSE (0)。

```
if(x<0 || x>=max_bars) Print("超出范围");
```

## 逻辑运算符 AND (&&)

x和y值的逻辑运算符 AND (&&). 如果x和y值为真的(not null)，表达式值为TRUE (1)。否则，值为FALSE (0)。

## 布尔运算的摘要评估

所谓的 "摘要评估"模式是应用到布尔型操作系统中的， i.e.当表达式可以被精确的终止时，该表达计算就会停止。

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 简单判断的第一示例
   if(func_false() && func_true())
     {
      Print("Operation &&: You will never see this expression");
     }
   else
     {
      Print("Operation &&: Result of the first expression is false, so the second wasn
     }
//--- 简单判断的第二示例
   if(!func_false() || !func_true())
     {
      Print("Operation ||: Result of the first expression is true, so the second wasn'
     }
   else
     {
      Print("Operation ||: You will never see this expression");
     }
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 函数总是返回false                                                |
//+------------------------------------------------------------------+
bool func_false()
  {
   Print("Function func_false()");
   return(false);
  }
//+------------------------------------------------------------------+
//| 函数总是返回 true                                                |
//+------------------------------------------------------------------+
bool func_true()
  {
   Print("Function func_true()");
   return(true);
  }
```

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 逐位运算

## 补码

补充变量值，表达值包含1，可变值包含0，表达值包含0，可变值包含1。

```
b = ~n;
```

示例：

```
   char a='a',b;
   b=~a;
   Print("a = ",a, "  b = ",b);
// 结果将会是:
// a = 97   b = -98
```

## 右移

运算符x向右移动到数字y代表二进制代码。如果移动的值是无符号类型，进行逻辑右移，即左侧将被零填满。

如果移动的值是符号类型，进行算术右移，即左侧将被符号填满 (如果数字是正值，符号为零值；如果数字为负值，符号值为1)。

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

示例：

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- 右移
   b=a>>1;
   Print("After:   a = ",a, "  b = ",b);
// 结果会是:
// 之前:  a = 97   b = 98
// 以后:   a = 97   b = 48
```

## 左移

运算符x向左移动到数字y代表二进制代码，即右侧将被零填满。

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

示例：

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- 左移
   b=a<<1;
   Print("After:   a = ",a, "  b = ",b);
// 结果会是:
// 之前:  a = 97   b = 98
// 之后:   a = 97   b = -62
```

© 2000-2025, MetaQuotes Ltd.

不建议移动大的数字或者与移动变量长度相等的数字，因为运算结果无法确定。

## 位逻辑运算符 AND

二进制的x和y代表位逻辑运算符AND。在所有数组中x和y的值都不含有零表达式的值包含1(TRUE)；在所有其他数字中包含 0 (FALSE)。

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

示例：

```
   char a='a',b='b';
//--- AND 操作
   char c=a&b;
   Print("a = ",a,"  b = ",b);
   Print("a & b = ",c);
// 结果会是:
// a = 97   b = 98
// a & b = 96
```

## 位逻辑运算符OR

二进制的x和y代表位逻辑运算符OR。在所有数字中x和y的值都不等于零表达值包含1 并且在所有其他数字中包含0。

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

示例：

```
   char a='a',b='b';
//--- 或者操作
   char c=a|b;
   Print("a = ",a,"  b = ",b);
   Print("a | b = ",c);
// 结果将会是:
// a = 97   b = 98
// a | b = 99
```

## 位逻辑运算符Exclusive

二进制的x 和y代表位逻辑运算符EXCLUSIVE (eXclusive OR)。在所有数字中x和y的值都不同于二进制值表达值包含1并且在所有其他数字中包含0 。

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

示例：

```
   char a='a', b='b';
//--- 免除或者执行
   char c=a^b;
   Print("a = ",a,"  b = ",b);
   Print("a ^ b = ",c);
// 结果会是:
```

© 2000-2025, MetaQuotes Ltd.

```
// a = 97   b = 98
// a ^ b = 3
```

位逻辑运算符只作用于integers。

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 其他运算

## 指数（[]）

在数组第一元素的位置，表达式值为i的系列数变量值。

示例：

```
array[i] = 3; // 数组的3的计算值到第i个元素。
```

只有整数能够成为数组指数。四维以下的数组是禁止的。每组的检测是从0到测量大小-1。 特定情况下，对于维数组由50 个元素组成，参照的第一个数组将为[0]，这样最后一个数组将是[49]。

访问超出数组，正在执行的子系统生成危险错误，程序停止。

## 调用x1, x2 ,..., xn自变数函数

每一个自变数可以显示一个常数，一个变量和相应类型表达式。自变数的通过必须根据通道命令。

用此函数返回表达式值。如果返回的表达式值为空，一些函数不能进行中转。注意表达式按照这个顺序执行expressions x1,..., xn 。

示例：

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

## 逗号操作符（,）

从左到右的表达式用逗号分开。所有表达式的计算是从左至右的。结果类型和值相互吻合，说明表达式是正确的。参量列表可以作为范例被通过 (参阅上文)。

示例：

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

## 点操作符（.）

对于直接访问结构和类的公共成员 时，使用点操作符。语法：

© 2000-2025, MetaQuotes Ltd.

```
Variable_name_of_structure_type.Member_name
```

示例：

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

## 范围解析操作符（::）

mql5中每个函数都有其执行范围。例如，Print()系统函数可以在全局范围使用。Imported函数导入时使用。classes函数用在类的范围。范围解析操作语法如下：

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

如果没有范围名，就属于全局范围函数。如果没有范围解析操作，可以在最近范围内找到函数。如果本地范围没有函数n，需要全局搜寻函数。

范围解析操作也用在 确定函数-类成员。

```
type Class_name::Function_name(parameters_description)
   {
// 函数主体
   }
```

在一个程序使用不同执行上下文的几个同名函数可能会引起歧义。无明确范围规范的函数调用的优先顺序如下：

1. 类方法。如果在类中设置了无指定名称的函数，那么移到下一级别。
2. MQL5函数。如果语言没有这种函数，那么移到下一级别。
3. 用户定义全局函数。如果没有找到有指定名称的函数，那么移到下一级别。
4. 导入函数。如果没有发现有指定名称的函数，那么编译器返回一个错误。

若要避免函数调用歧义，就要使用范围解析操作，始终明确指定函数范围。

示例：

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

© 2000-2025, MetaQuotes Ltd.

```
class CCheckContext
  {
   int          m_id;
public:
                CCheckContext() { m_id=1234; }
protected:
   int          GetLastError() { return(m_id); }
  };
class CCheckContext2 : public CCheckContext
  {
   int          m_id2;
public:
                CCheckContext2() { m_id2=5678; }
   void         Print();
protected:
   int          GetLastError() { return(m_id2); }
  };
void CCheckContext2::Print()
  {
   ::Print("Terminal GetLastError",::GetLastError());
   ::Print("kernel32 GetLastError",kernel32::GetLastError());
   ::Print("parent GetLastError",CCheckContext::GetLastError());
   ::Print("our GetLastError",GetLastError());
  }
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CCheckContext2 test;
   test.Print();
  }
//+------------------------------------------------------------------+
```

## 获得数据类型大小或者任何类型数据对象大小的运算（sizeof）

使用sizeof运算，标识符或者类型大小可以确定。sizeof运算有以下格式：

示例：

```
sizeof(expression)
```

任何标识符，或者类型名都在括号中，可用作表达式。注意空类型名不可以使用，标识符不属于二进制，也不是函数名。

如果表达式为静态数组名（例如给出第一尺寸），那么结果就是整个数组大小（例如类型长度产品）。如果表达式是动态数组名（没有确定第一尺寸），结果就是动态数组对象大小。

© 2000-2025, MetaQuotes Ltd.

当sizeof用在结构或者类型名时，或者结构或者类型标识符时，结果就是结构或者类的真实大小。

示例：

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

在编辑步骤计算大小。

另见

优先规则

© 2000-2025, MetaQuotes Ltd.

# 优先规则

下面是从上到下的运算优先规则，优先级高的将先被运算。

注意：MQL5语言运算优先依据C++优先规则，不同于MQL4语言。

| 运算 | 描述 | 执行顺序 |
| --- | --- | --- |
| ()<br>[]<br>. | 函数调用<br>数组元素参考<br>引用结构元素 | 从左到右 |
| !<br>~<br>-<br>++<br>--<br>(type)<br>sizeof | 真假运算符<br>位逻辑运算符（补码）<br>改变运算符<br>增量<br>减量<br>类型转换<br>决定字节大小 | 从右到左 |
| *<br>/<br>% | 乘法<br>除法<br>百分比 | 从左到右 |
| +<br>- | 加法<br>减法 | 从左到右 |
| <<<br>>> | 左移<br>右移 | 从左到右 |
| <<br><=<br>><br>>= | 小于<br>小于等于<br>大于<br>大于等于 | 从左到右 |
| ==<br>!= | 等于<br>不等于 | 从左到右 |
| & | 位逻辑运算符AND | 从左到右 |
| ^ | 位逻辑运算符 OR | 从左到右 |
| \| | 位逻辑运算符 OR | 从左到右 |
| && | 逻辑AND | 从左到右 |
| \|\| | 逻辑OR | 从左到右 |
| ?: | 假设运算 | 从右到左 |
| =<br>*=<br>/=<br>%=<br>+=<br>-= | 值<br>乘法值<br>除法值<br>百分比值<br>加法值<br>减法值 | 从右到左 |

© 2000-2025, MetaQuotes Ltd.

| 运算 | 描述 | 执行顺序 |
| --- | --- | --- |
| <<=<br>>>=<br>&=<br>^=<br>\|= | 左移值<br>右移值<br>位逻辑运算符AND值<br>位逻辑运算符OR值<br>位逻辑运算符OR值 | |
| , | 逗号 | 从左到右 |

若要改变运算操作顺序，使用更高一级的圆括号。

© 2000-2025, MetaQuotes Ltd.

# 操作符

语言操作符必须对执行完成任务的一些运算法操作进行描述。程序本身是这样的序列语句。语句逐个随后以分号分离。

| 操作符 | 描述 |
| --- | --- |
| Compound operator {} | 花括号 {}括起来的一个或者多个任一类型操作符 |
| Expression operator (;) | 分号 (;)结尾的表达式 |
| return 操作符 | 终止当前函数和返回控制访问程序 |
| if-else 假设操作符 | 需要作出选择时使用 |
| ?: 假设操作符 | if-else假设操作符的简单类似物 |
| switch 选择操作符 | 控制与表达式值一致的操作符 |
| while 循环操作符 | 执行操作符直到表达式变成错误。每次重复前都要检查表达式。 |
| for 循环操作符 | 执行操作符直到表达式变成错误。每次重复前都要检查表达式。 |
| do-while 循环操作符 | 执行操作符直到表达式变成错误。每次循环后进行检测。循环主体至少执行一次。 |
| break 操作符 | 终止执行最近的外部操作符，while，do-while 或者 for |
| continue 操作符 | 控制最近的外部循环操作符 while，do-while 或者for的起点。 |
| 矩阵乘法运算符 @ | @ MQL5 运算符根据线性代数规则实现矩阵乘法。它允许将矩阵和向量相乘，以及执行向量的标量乘法。 |
| new 操作符 | 创建大小合适的对象以及返回创建对象的描述符。 |
| delete 操作符 | 删除新操作符创建的对象 |

一个语句能占领一条或几条线。二个或更多语句可能位于同样线。控制执行顺序的语句(if, if-else, switch, while and for) 可以相互插入。

示例：

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

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 复合操作符

一个复合操作符 (a block)有一个(一个区段)和由一个或多个任何类型的操作符组成的的附件{}. 每个表达式使用分号作为结束(;)。

示例：

```
if(x==0)
  {
   Print("无效位置 x = ",x);
   return;
  }
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 表达式操作符

任何以分号(;)结束的表达式都被视为是一个操作符。这里是一些表达式操作符得范例:

## 赋值操作符

标识符 =表达式;

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

赋值运算符可以在一个表达式中运用多次。在这种情况下,表达式处理从右向左。

## 函数调用操作符

Function_name (argument1,..., argumentN);

```
FileClose(file);
```

## 空操作符

它是由分号(;)组成并且使用在一个检测操作符中。

另见

变量初始化, 可见范围和变量使用期, 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 返回操作符

返回操作符终止当前函数操作，返回控制访问程序。表达式计算结果返回调用函数。表达式可以包括返回操作符。

示例：

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

函数中，空返回类型函数，必须使用无表达式的return操作符：

```
void SomeFunction()
  {
   Print("Hello!");
   return;     // 操作符可以被移除
  }
```

函数右括号意思是无表达式返回操作符的隐性执行。

能返回的有：简单类型，简单结构，目标指针。*返回*操作符您不能返回任何数组，类对象，复合结构类型变量。

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# If-Else 假设操作符

需要选择时使用IF-ELSE操作符。语法形式如下：

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

如果表达式是真实的，操作符1执行，并按照操作符2给出的检测（操作符2不执行）。如果表达式是错误的，操作符2被执行。

if操作符else部分可能被忽略。if操作符忽略else部分，显示分歧可能会嵌入。这种情况下，else位置在先前if操作符的最近部位，这样不会出现else部分。

示例：

```
//--- else 部分提及到第二个if操作符:
if(x>1)
   if(y==2) z=5;
else      z=6;
//--- else部分提及到第一个if操作符:
if(x>1)
  {
   if(y==2) z=5;
  }
else         z=6;
//--- 嵌入操作符
if(x=='a')
  {
   y=1;
  }
else if(x=='b')
  {
   y=2;
   z=3;
  }
else if(x=='c')
  {
   y=4;
  }
else Print("ERROR");
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 假设操作符?:

三进制操作符一般形式如下：

```
expression1? expression2:expression3
```

对于第一个操作-"expression1"-任何表达式在布尔型中值都可以用。如果结果是true，那么第二个操作的操作符，例如 "expression2"，就被执行。

如果第一操作是false，第三操作-"expression3"-就被执行。第二和第三操作，例如 "expression2"和"expression3"应该返回第一类型值，不应该是空类型。执行假设操作符的结果就是expression2的结果或者expression3的结果，都取决于 expression1。

```
//--- 一天中开盘价和收盘价的标准化不同点
double true_range = (High==Low)?0:(Close-Open)/(High-Low);
```

输入等同于如下：

```
   double true_range;
   if(High==Low)true_range=0;                 // 如果最高价等于
   else true_range=(Close-Open)/(High-Low); // 如果范围无效
```

# 操作符使用限制

基于"expression1"值，操作符必须返回两个值中的一个 - 或者"expression2"或者"expression3"。对这些表达式有几个限制：

1. 不要混淆 简单类型 或 枚举类型 的用户定义类型。NULL 可能被用于指针.
2. 如果值是简单类型，操作符将会是最大类型 (请见类型转换 )。
3. 如果一个值是枚举类型而第二个值是数字类型，那么枚举类型被整型所取代并实施第二个规则。
4. 如果两个值都是枚举类型，它们的类型必须完全相同，并且操作符将是枚举类型。

限制用户定义类型 (类或结构)：

a) 类型必须完全相同或者一个应该源自另一个。

b) 如果类型不相同 (继承类型)，那么子类型会隐式转换至父类型，例如操作符将成为父类型。.

c) 不要混淆对象和指针 - 两个表达式都是或是对象或是 指针。NULL 不能用于指针。

注意

谨慎使用条件操作符当做重载函数的论据，因为条件操作符的结果类型在程序编译时定义。并且该类型定义为较大的"expression2"和"expression3"。

示例：

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

bool   Expression1=true;
double Expression2=M_PI;
```

© 2000-2025, MetaQuotes Ltd.

```
string Expression3="3.1415926";

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

   func(Expression1?Expression2:Expression3);   // 隐式转换到字符串的警告
   func(!Expression1?Expression2:Expression3);  // 隐式转换到字符串的警告
  }

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

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 循环操作符While

while操作符是由一个已检测的表达式和操作符构成，需要满足以下条件：

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

如果表达式为true，操作符执行直至表达式变成false。如果表达式为false，将检测最近操作符。在操作符执行前，一个表达式值已经被指定。不过，如果开始表达式为 false，操作符根本不会执行。

注意

如果预计一个循环中处理大量的迭代，建议您检查使用IsStopped()函数被迫中止程序的事实。

示例

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

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 切换操作符

在*case*全部变量和相应表达式值检测的操作符之内比较常数表达式值。每一个*case*变量会在整数常数表达式内标注。常数表达式不包含函数变量调用。*switch*表达式操作符必须是整数类型 - int 或 uint。

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

如果在*case*操作符等于表达式值，操作符*default*标签连接将会执行。此*default*变量无需在最后。如果相应表达式值和*default*变量没有获取，不会有任何执行。

关键词*case*和常数被标注，并且if 操作符执行 *case*变量，程序将执行以下所有操作符直至*break*操作符生成。

一个常数表达式的计算是在编译期间。在一个*switch*操作符内部存在两个相同值的常数。

示例：

```
//--- 第一示例
switch(x)
  {
   case 'A':
      Print("CASE A");
      break;
   case 'B':
   case 'C':
      Print("CASE B or C");
      break;
   default:
      Print("NOT A, B or C");
      break;
  }

//---  第二示例
   string res="";
   int i=0;
   switch(i)
     {
      case 1:
         res=i;break;
      default:
         res="default";break;
      case 2:
         res=i;break;
      case 3:
```

© 2000-2025, MetaQuotes Ltd.

```
            res=i;break;
        }
    Print(res);
/*
    结果
    默认
*/
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 循环操作符For

for操作符由三个表达式和一个执行操作符组成：

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

用表达式1Expression来定义初始变量,当表达式2Expression2 为真的时候执行操作运算符for，在每次循环结束后执行表达式3 Expression3。如果true， 运算符 for 将被执行。循环重复直至Expression2变为false。如果false，循环将会被中断并且检测运算符文本。稍候执行。

此 for运算符下列运算符成功：

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

for操作符中可以缺少任何三个或者全部三个表达式，但是分隔它们的分号 (;) 必须省略。如果表达式2省略，则意味着不变的true。for(;;)操作符是持续循环，与 while(1)操作符一样。每一个表达式1或者3都由几个逗号 ','组成的表达式构成。

注意

如果预计一个循环中处理大量的迭代，建议您检查使用IsStopped()函数被迫中止程序的事实。

示例：

```
for(x=1;x<=7000; x++)
  {
   if(IsStopped())
      break;
   Print(MathPower(x,2));
  }
//--- 另一个示例
for(;!IsStopped();)
  {
   Print(MathPower(x,2));
   x++;
   if(x>10) break;
  }
//--- 第三示例
for(i=0,j=n-1;i<n && !IsStopped();i++,j--) a[i]=a[j];
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 循环操作符 do while

for 和 while 从起点循环检测终止，不在循环末端。第三种循环操作符do - whileo每次循环重复后，在最后检测终止状态。循环主体至少执行一次。

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

首先执行操作符，然后计算表达式。如果是true，那么操作符再次执行。如果表达式变成false，循环终止。

注意

如果预计一个循环中处理大量的迭代，建议您检查使用IsStopped()函数被迫中止程序的事实。

示例：

```
//--- 计算斐波纳契数列
   int counterFibonacci=15;
   int i=0,first=0,second=1;
   int currentFibonacciNumber;
   do
     {
      currentFibonacciNumber=first+second;
      Print("i = ",i,"  currentFibonacciNumber = ",currentFibonacciNumber);
      first=second;
      second=currentFibonacciNumber;
      i++; // 没有这个操作符会出现一个无限循环!
     }
   while(i<counterFibonacci && !IsStopped());
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 嵌入操作符

break操作符终止最近外部嵌入操作符 switch, while, do-while 或者 for 的执行。在终止操作符之后给出检测操作符。这个操作符的目的之一：当中心值指定为变量时，操作符完成循环执行。

示例：

```
//--- 搜索第一个零元素
for(i=0;i<array_size;i++)
  if(array[i]==0)
    break;
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 继续操作符

一个继续操作符。 我们将其放在嵌套内的指定位置,用来在指定情况下跳过接下来的运算，直接跳入下一次的循环 while, do-while 或者 for操作符。操作符break位置与此操作符相反。

示例：

```
//--- 所有非零元素总和
int func(int array[])
  {
   int array_size=ArraySize(array);
   int sum=0;
   for(int i=0;i<array_size; i++)
     {
      if(a[i]==0) continue;
      sum+=a[i];
     }
   return(sum);
  }
```

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 矩阵乘法运算符 @

@ MQL5 运算符根据线性代数规则实现矩阵乘法。它允许将矩阵和向量相乘，以及执行向量的标量乘法。

支持元素类型：

- float
- double
- complex<float>
- complex<double>

重要：左操作数和右操作数中的元素类型必须匹配。

使用示例

1. 矩阵乘法（矩阵×矩阵）

```
matrix A(2, 3);
matrix B(3, 2);
matrix C = A @ B; // 结果:大小为 [2,2] 的矩阵 C
```

2. 矩阵乘法（矩阵×向量）

```
matrix M(2, 3);
vector V(3);
vector R = M @ V; // 结果:2 个元素的向量 R
```

3. 矩阵乘法（向量x矩阵）

```
matrix M(2, 3);
vector V(1, 2);
vector R = V @ M; // 结果:3 个元素的向量 R
```

4. 标量乘法（向量×向量）

```
vector V1(1, 3), V2(1, 3);
double r = V1 @ V2; // 结果:标量
```

请注意

维度必须符合乘法规则：第一个操作数中的列数 = 第二个操作数的行数。

如果发生维度错误，将在运行时触发异常。

左边的向量被认为是水平的（1×n）。

© 2000-2025, MetaQuotes Ltd.

右边的向量被认为是垂直的 (n×1)。

@ 运算的优先级对应于乘法的优先级，即对于 D=C+A@B 的条目，首先执行矩阵乘法 T=A@B，然后进行元素加法 D=C+T。

参见

MatMul, GeMM, Kron, Dot,

© 2000-2025, MetaQuotes Ltd.

# 对象创建操作符new

new 操作符自动创建一个相应大小的对象，称为对象构造函数并回转已经创建的对象描述符。失败的情况下，操作符返回一个与常量NULL相比较的null描述符。

new操作符仅能用于类对象。不能应用于结构。

操作符不用于创建对象数组。若要做这个，使用 ArrayResize() 。

示例：

```
//+------------------------------------------------------------------+
//| 图形创建                                                         |
//+------------------------------------------------------------------+
void CTetrisField::NewShape()
  {
   m_ypos=HORZ_BORDER;
//--- 随机创建7个可能形状中的一个
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- 绘画
   if(m_shape!=NULL)
     {
      //--- 预置
      m_shape.SetRightBorder(WIDTH_IN_PIXELS+VERT_BORDER);
      m_shape.SetYPos(m_ypos);
      m_shape.SetXPos(VERT_BORDER+SHAPE_SIZE*8);
      //--- 绘画
      m_shape.Draw();
     }
//---
  }
```

注意对象描述符不是内存指标。

用new操作符创建的对象不能用delete操作符直接移动。

另见

变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 对象删除操作符 delete

delete操作符删除通过new操作符创建的对象，称为相关的类析构函数并释放由对象占据的内存。现存对象的真实析构函数用作操作对象。delete操作执行后，对象析构函数无效 。

示例：

```
      //--- 删除图形
      delete m_shape;
      m_shape=NULL;
      //--- 创建一个新图形
      NewShape();
```

另见

变量初始化， 可见范围和变量使用期， 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 函数

每个任务都可以分解成子任务，既可以用代码形式直接表示，也可以分成子任务。这个方法称为*逐步求精*。函数用来输入要解决的子任务的代码。描述函数的代码称为*函数定义*:

```
function_header
  {
   instructions
  }
```

第一个括号前是函数定义的表头，括号中间是函数定义主体。函数表头包括返回值类型描述，名称(标识符)和正式参数 。通过函数的参数有限制，不能超过64。

函数不能从程序的其他部分多次调用。实际上，返回类型，函数标识符和参数类型组成*函数原型*。

函数原型是函数说明，但是不是定义。由于明确说明返回类型和自变量类型列表，调用函数时严格检测类型和隐藏的类型转换。经常使用函数说明提高代码可读性。

函数定义必须与其说明相匹配。每个说明的函数都要下定义。

示例：

```
double                          // 返回值类型
linfunc (double a, double b) // 函数名和参量列表
  {
                                // 组合操作符
   return (a + b);              // 返回值
  }
```

返回操作符能够返回位于这个操作符的表达式的值。如果有必要，表达式值可以转换为函数结果类型。返回到零值的函数描述为空型。

示例：

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

通过函数的参量可能存在由特定类型常数指定的默认值。

示例：

```
int somefunc(double a,
             double d=0.0001,
             int n=5,
             bool b=true,
             string s="passed string")
  {
   Print("Required parameter a = ",a);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("Pass the following parameters: d = ",d," n = ",n," b = ",b," s = ",s);
   return(0);
  }
```

如果任何参数有默认值，所有的子参数也一定有默认值。

错误说明示例：

```
int somefunc(double a,
             double d=0.0001,     // 声明默认值
             int n,               // 没有指定默认值 !
             bool b,              // 没有指定默认值 !
             string s="passed string")
  {
  }
```

另见

重载, 虚拟函数, 多态

© 2000-2025, MetaQuotes Ltd.

# 调用函数

如果显示的文件没有描述，它将考虑上下文的联系作为函数名称。

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

自变数（形式参数）以值的形式通过。计算每一个表达式 x1, . . . , xn 并将其值发送到函数。表达式计算命令值是被保证的。在执行系统测试数字和自变数类型期间会给出函数。这种形式的函数调用被称作调用值。

调用函数是一个通过函数返回的表达式的值。描述函数类型必须相应类型返回的值。全球范围内程序的任何一个部分函数是被声明或描述的，即其他函数之外。在另外函数里，函数不能被声明或描述。

示例：

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

函数的调用是默认参量，通过参量的列表是被限定的，但不是之前的第一默认参量。

示例：

```
void somefunc(double init,
              double sec=0.0001, //设置默认值
              int level=10);
//...
somefunc();                       // 错误调用。必须存在第一参量
somefunc(3.14);                   // 正确调用
somefunc(3.14,0.0002);            // 正确调用
somefunc(3.14,0.0002,10);         // 正确调用
```

当我们调用一个函数时，不可以忽略参量，存在默认值：

```
somefunc(3.14, , 10);             // 错误调用 -> 跳过第二参量
```

另见

重载, 虚拟函数, 多态

© 2000-2025, MetaQuotes Ltd.

# 传递函数

有两类函数，计算机语言可以用其传送自变量到子程序（函数）。第一类函数是通过值发送参数。这个函数将自变量值复制到形式函数参数。因此，函数中这个参数的任何改变都不会对自变量的相应调用产生影响。

```
//+------------------------------------------------------------------+
//| 通过值传递参数                                                   |
//+------------------------------------------------------------------+
double FirstMethod(int i,int j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=FirstMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//--- 执行脚本结果
//  调用前a 和 b : 14 8
//  调用后a 和 b : 14 8
```

第二类函数是通过引用传送。在这种情况下，参数引用（不是它的值）被传送到函数参数。函数内，被用来连接调用指定的实际参数。这意味着参数的改变将会影响用来调用函数的自变量。

```
//+------------------------------------------------------------------+
//| 通过引用传递参数                                                 |
//+------------------------------------------------------------------+
double SecondMethod(int &i,int &j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=SecondMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//+------------------------------------------------------------------+
//--- 执行脚本结果
//  调用前a 和 b: 14 8
//  调用后a 和 b: 28 4
```

MQL5使用这两种函数，有一个例外就是：数组和结构类型变量（类的对象）总是通过引用传送。为了在真实参数中排除改变（函数调用传送的自变量）使用常数 访问说明符。若尝试改变说明*常量*说明符的变量，编译程序会生成错误。

## 注解

注意用倒序传送参数到函数，例如，首先是最近的函数被计算和传送，然后倒数第二。最后计算和传送的参数是括号后的第一个。

示例：

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

   func(a[i],a[i++],"First call (i = "+string(i)+")");
   func(a[i++],a[i],"Second call (i = "+string(i)+")");
// 结果:
// 首先调用 (i = 0) : par1 = 1     par2 = 0
// 第二调用 (i = 1) : par1 = 1     par2 = 1

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

首先上面的示例中先调用变量*i*参与合并字符串。

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

© 2000-2025, MetaQuotes Ltd.

这里值不可更改。变量 ***i*** 加入***a[i++]***数组的运算，例如加入数组元素i，变量***i***就会增加。之后，仅计算变量 ***i*** 改变值第一个参数。

第二步，计算所有三个参数，使用调用函数第一步计算的和i的同值，仅计算第一参数后变量***i***会再次更改。

另见

可见范围和变量使用期，重载，虚拟函数，多态

© 2000-2025, MetaQuotes Ltd.

# 重载函数

通常函数名反映它的作用。一般说来，可读程序包括各种挑选出来的标识符 。有时不同的函数用于同样的作用。例如，计算双精度数数组的平均值的函数，也会操作整数数组。都可以称为数组平均数AverageFrom Array：

```
//+------------------------------------------------------------------+
//| 双精度类型数组的平均数计算                                       |
//+------------------------------------------------------------------+
double AverageFromArray(const double array[],int size)
  {
   if(size<=0) return 0.0;
   double sum=0.0;
   double aver;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];     // 添加到双精度型
     }
   aver=sum/size;        // 正好总数被数字除
//---
   Print("Calculation of the average for an array of double type");
   return aver;
  }
//+------------------------------------------------------------------+
//| 整型数组的平均数计算                                             |
//+------------------------------------------------------------------+
double AverageFromArray(const int array[],int size)
  {
   if(size<=0) return 0.0;
   double aver=0.0;
   int sum=0;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];     // 添加到双精度
     }
   aver=(double)sum/size;// 给予双精度类型总值，并且相除
//---
   Print("Calculation of the average for an array of int type");
   return aver;
  }
```

每个函数通过 Print()函数输出信息：

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

程序选择与自变量类型及其数量一致的必需的函数。选择依据的规则，称为*署名匹配算法*。署名就是用于说明函数类型的列表。

© 2000-2025, MetaQuotes Ltd.

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int    a[5]={1,2,3,4,5};
   double b[5]={1.1,2.2,3.3,4.4,5.5};
   double int_aver=AverageFromArray(a,5);
   double double_aver=AverageFromArray(b,5);
   Print("int_aver = ",int_aver,"   double_aver = ",double_aver);
  }
//--- 脚本结果
// 为整型数组计算平均值
// 为双精度型数组计算平均值
// 整型 平均值= 3.00000000     双精度 平均值= 3.30000000
```

重载就是一个函数定义多值。根据函数接收的自变量类型选择特定的值。基于调用函数时对应的自变量列表选择特定的函数，到函数说明参数列表。

当调用重载的函数，程序必需有一个运算法则选择合适的函数。该算法是依据现有类型的转化完成这个选择。最佳关联是唯一的。至少对于一个自变量来说，是最好的，对其他相关的自变量也是一样。

以下就是每个自变量匹配的算法。

## 选择一个重载函数的运算法则

1. 匹配严格（尽可能）。
2. 尝试标准类型提高。
3. 尝试标准类型转换。

标准类型提高好于其他标准转换。提高就是浮点型向双精度型转换，布尔型，字符型，短整型或者计数向整型转换。类似整型数组的类型转换也属于类型转换。相似类型有：布尔型，字符型，无符号字符型，这三种都是单字节整数；双字节整数是短整型和无符号短整型；4字节整数是整型，无字符整型，和颜色；长整型，无符号长整型，和日期时间。

当然，匹配严格是最好的。若要达到这样的一致性，就要使用类型转换。编译程序不能处理含糊状况。因此不能依靠类型的微妙不同点和使重载函数不明确的隐式变换。

如果怀疑，请使用显式转换确保严格相符。

MQL5中重载的函数可以在ArrayInitialize()函数示例中看到。

函数重载规则用于类函数重载。

允许系统函数重载，但是要注意到编译程序可以准确的选择必要函数。例如，用三种不同方法重载系统函数 fmax()，但是仅有两个变量是正确的。

示例：

© 2000-2025, MetaQuotes Ltd.

```
// 允许重载-在参数数字上不同
double fmax(double a,double b,double c);

// 错误重载
// 参数是不同的，但是最后一个有默认值
// 这就导致系统函数的隐藏，不可接受调用
double fmax(double a,double b,double c=DBL_MIN);

// 通过参数类型正常重载
int fmax(int a,int b);
```

另见

重载，虚拟函数，多态

© 2000-2025, MetaQuotes Ltd.

# 操作重载

为了便于代码读写，允许重载一些操作。使用关键字operator编写重载操作符。下面这些操作符可以重载：

- 二进制 +,-,/,*,%,<<,>>,==,!=,<,>,<=,>=,=,+=,-=,/=,*=,%=,&=,|=,^=,<<=,>>=,&&,||,&,|,^
- 一进制 +,-,++,--,!,~
- 赋值操作符 =
- 索引操作符 []

操作重载允许使用操作记号（以简单的表达式书写）用于复杂的对象 - 结构和类。使用重载操作书写表达式简化了源代码视图，因为隐藏了更复杂的实施。

例如，考虑复数，其中包括实数和虚数。它们广泛用于算术学。MQL5语言没有表示复数的数据类型，但是可以用结构或类的形式创建新的数据类型。声明复杂结构并定义四种实施四个算术运算的方法：

```
//+------------------------------------------------------------------+
//|  复数操作结构                                                    |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // 实数
   double            im; // 虚数
   //--- 构造函数
                     complex():re(0.0),im(0.0) {  }
                     complex(const double r):re(r),im(0.0) {  }
                     complex(const double r,const double i):re(r),im(i) {  }
                     complex(const complex &o):re(o.re),im(o.im) { }
   //--- 算术运算
   complex           Add(const complex &l,const complex &r) const;  // 加法
   complex           Sub(const complex &l,const complex &r) const;  // 减法
   complex           Mul(const complex &l,const complex &r) const;  // 乘法
   complex           Div(const complex &l,const complex &r) const;  // 除法
  };
```

现在，在我们的代码中我们可以声明表示复数的变量，并使用它们。

例如：

```
void OnStart()
  {
//--- 声明并初始化复杂类型的变量
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- 两个数字总和
   complex z;
   z=a.Add(a,b);
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 两个数字相乘
   z=a.Mul(a,b);
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- 两个数字相除
   z=a.Div(a,b);
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
```

但是对于复数的普通算术操作，使用一般的操作符"+", "-", "*"和 "/"会更加方便。

关键字operator用于定义执行类型转换的成员函数。类对象变量的一进制和二进制操作可以作为非静态成员函数重载。它们在类对象上隐式操作。

大多数二进制操作可以像普通函数一样重载，采取一个或两个参数作为类变量或该类对象的指针。为了类型复杂，声明重载类似如下：

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

所有脚本示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明和初始化复杂类型的变量
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
   //a.re=5;
   //a.im=1;
   //b.re=-1;
   //b.im=-5;
//--- 两个数字总和
   complex z=a+b;
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- 两个数字相乘

   z=a*b;
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- 两个数字相除
   z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
//+------------------------------------------------------------------+
//| 复数操作结构                                                     |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // 实数
   double            im; // 虚数
   //--- 构造函数
                     complex():re(0.0),im(0.0) {  }
                     complex(const double r):re(r),im(0.0) {  }
                     complex(const double r,const double i):re(r),im(i) {  }
                     complex(const complex &o):re(o.re),im(o.im) { }
   //--- 算术运算
   complex           Add(const complex &l,const complex &r) const;  // 加法
   complex           Sub(const complex &l,const complex &r) const;  // 减法
   complex           Mul(const complex &l,const complex &r) const;  // 乘法
   complex           Div(const complex &l,const complex &r) const;  // 除法
   //--- 二进制操作符
   complex operator+(const complex &r) const { return(Add(this,r)); }
   complex operator-(const complex &r) const { return(Sub(this,r)); }
   complex operator*(const complex &r) const { return(Mul(this,r)); }
   complex operator/(const complex &r) const { return(Div(this,r)); }
  };
//+------------------------------------------------------------------+
//| 加法                                                             |
//+------------------------------------------------------------------+
complex complex::Add(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re+r.re;
   res.im=l.im+r.im;
//--- 结果
   return res;
  }
//+------------------------------------------------------------------+
//| 减法                                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
complex complex::Sub(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re-r.re;
   res.im=l.im-r.im;
//--- 结果
   return res;
  }
//+------------------------------------------------------------------+
//| 乘法                                                             |
//+------------------------------------------------------------------+
complex complex::Mul(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re*r.re-l.im*r.im;
   res.im=l.re*r.im+l.im*r.re;
//--- 结果
   return res;
  }
//+------------------------------------------------------------------+
//| 除法                                                             |
//+------------------------------------------------------------------+
complex complex::Div(const complex &l,const complex &r) const
  {
//--- 空复数
   complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- 检查零
   if(r.re==0 && r.im==0)
     {
      Print(__FUNCTION__+": number is zero");
      return(res);
     }
//--- 辅助变量
   double e;
   double f;
//--- 选择计算变量
   if(MathAbs(r.im)<MathAbs(r.re))
     {
      e = r.im/r.re;
      f = r.re+r.im*e;
      res.re=(l.re+l.im*e)/f;
      res.im=(l.im-l.re*e)/f;
     }
   else
     {
      e = r.re/r.im;
      f = r.im+r.re*e;
      res.re=(l.im+l.re*e)/f;
      res.im=(-l.re+l.im*e)/f;
     }
//--- 结果
   return res;
  }
```

大部分一进制的类操作可以作为接受单类对象参数或其指针的普通函数进行重载。添加重载一进制操作 "-" 和 "!"。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 复数操作结构                                                     |
//+------------------------------------------------------------------+
struct complex
  {
   double            re;        // 实数
   double            im;        // 虚数
...
   //--- 一进制操作符
   complex operator-()  const; // 一进制减操作
   bool    operator!()  const; // 非操作
  };
...
//+------------------------------------------------------------------+
//| 重载"一进制减" 操作符                                            |
//+------------------------------------------------------------------+
complex complex::operator-() const
  {
   complex res;
//---
   res.re=-re;
   res.im=-im;
//--- 结果
   return res;
  }
//+------------------------------------------------------------------+
//| 重载 "逻辑非" 操作符                                             |
//+------------------------------------------------------------------+
bool complex::operator!() const
  {
//--- 复数的实数和虚数等于零吗?
   return (re!=0 && im!=0);
  }
```

现在我们可以检查零复数值并获得一个负值：

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明和初始化复杂类型的变量
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- 两个数字相除
   complex z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//--- 复数默认等于零 默认构造函数re==0 和 im==0)
   complex zero;
   Print("!zero=",!zero);
//--- 分配负值
   zero=-z;
   PrintFormat("z=%.2f+i*%.2f,  zero=%.2f+i*%.2f",z.re,z.im, zero.re,zero.im);
   PrintFormat("-zero=%.2f+i*%.2f",-zero.re,-zero.im);
//--- 再次检查零值
   Print("!zero=",!zero);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

注意我们无需重载赋值操作符 "=",因为 简单类型的结构 可以直接复制成另一个。因此,我们现在可以用普通方式书写计算涉及复数的代码。

重载索引操作符允许以简单熟悉的方式,获得对象内附有的数组值,它也有助于更好地阅读源代码。例如,我们需要在指定位置提供访问符号字符串。MQL5字符串是一个单独的类型字符串,它并不是符号数组,但是在重载索引操作的帮助下,我们可以在生成的CString类中提供简单透明的工作:

```
//+------------------------------------------------------------------+
//| 访问类似符号数组的字符串符号的类                                 |
//+------------------------------------------------------------------+
class CString
  {
   string            m_string;

public:
                     CString(string str=NULL):m_string(str) { }
   ushort operator[] (int x) { return(StringGetCharacter(m_string,x)); }
  };
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 从字符串接收符号的数组
   int     x[]={ 19,4,18,19,27,14,15,4,17,0,19,14,17,27,26,28,27,5,14,
                 17,27,2,11,0,18,18,27,29,30,19,17,8,13,6 };
   CString str("abcdefghijklmnopqrstuvwxyz[ ]CS");
   string  res;
//--- 使用来自str变量的符号组成短语
   for(int i=0,n=ArraySize(x);i<n;i++)
     {
      res+=ShortToString(str[x[i]]);
     }
//--- 显示结果
   Print(res);
  }
```

另一个重载索引操作的示例就是矩阵操作。矩阵表示一个二维动态数组,数组大小不会提前定义。因此,您不能声明没有指定第二维度大小的数组形式array[][],以及作为参数通过该数组。可能的解决方案就是一个特别的类CMatrix,它包含CRow 类对象数组。

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 矩阵的加法和乘法操作
   CMatrix A(3),B(3),C();
//--- 准备数组行
   double a1[3]={1,2,3}, a2[3]={2,3,1}, a3[3]={3,1,2};
   double b1[3]={3,2,1}, b2[3]={1,3,2}, b3[3]={2,1,3};
//--- 填补矩阵
   A[0]=a1; A[1]=a2; A[2]=a3;
   B[0]=b1; B[1]=b2; B[2]=b3;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在专家日志输出矩阵
   Print("---- Elements of matrix A");
   Print(A.String());
   Print("---- Elements of matrix B");
   Print(B.String());

//--- 矩阵加法
   Print("---- Addition of matrices A and B");
   C=A+B;
//--- 输出格式化字符串表示
   Print(C.String());

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

//--- 现在我们展示如何在动态数组matrix[i][j]风格下得到值
   Print("Output the values of matrix C elementwise");
//--- 通过矩阵行 - CRow 对象 - 循环
   for(int i=0;i<3;i++)
     {
      string com="| ";
      //--- 形成矩阵的行值
      for(int j=0;j<3;j++)
        {
         //--- 通过行数和列数获得矩阵元素
         double element=C[i][j];// [i] - 在数组 m_rows[]访问CRow ,
                                // [j] - 重载CRow索引操作符
         com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
        }
      com+="|";
      //--- 输出行值
      Print(com);
     }
  }
//+------------------------------------------------------------------+
//| 类"行"                                                           |
//+------------------------------------------------------------------+
class CRow
  {
private:
   double            m_array[];
public:
   //--- 构造函数和析构函数
                     CRow(void)           { ArrayResize(m_array,0);     }
                     CRow(const CRow &r)  { this=r;                     }
                     CRow(const double &array[]);
                    ~CRow(void){};
   //--- 行的元素数
```

© 2000-2025, MetaQuotes Ltd.

```
   int               Size(void) const    { return(ArraySize(m_array));}
   //--- 返回一个字符串值
   string            String(void) const;
   //--- 索引操作符
   double            operator[](int i) const  { return(m_array[i]);   }
   //--- 赋值操作符
   void              operator=(const double  &array[]); // 数组
   void              operator=(const CRow & r);         // 另一个 CRow 对象
   double            operator*(const CRow &o);          // CRow 对象乘法
  };
//+------------------------------------------------------------------+
//| 构造函数初始化数组行                                             |
//+------------------------------------------------------------------+
void  CRow::CRow(const double &array[])
  {
   int size=ArraySize(array);
//--- 如果数组不为空
   if(size>0)
     {
      ArrayResize(m_array,size);
      //--- 填值
      for(int i=0;i<size;i++)
         m_array[i]=array[i];
     }
//---
  }
//+------------------------------------------------------------------+
//| 数组赋值操作                                                     |
//+------------------------------------------------------------------+
void CRow::operator=(const double &array[])
  {
   int size=ArraySize(array);
   if(size==0) return;
//--- 填充数组值
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//|  CRow 赋值操作                                                   |
//+------------------------------------------------------------------+
void CRow::operator=(const CRow  &r)
  {
   int size=r.Size();
   if(size==0) return;
//--- 填充数组值
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=r[i];
//---
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 另一行乘法的操作符                                                 |
//+------------------------------------------------------------------+
double CRow::operator*(const CRow &o)
  {
   double res=0;
//--- 验证
   int size=Size();
   if(size!=o.Size() || size==0)
     {
      Print(__FUNCSIG__,": Failed to multiply two matrices, their sizes are different'
      return(res);
     }
//--- 乘以数组elementwise 并添加产品
   for(int i=0;i<size;i++)
      res+=m_array[i]*o[i];
//--- 结果
   return(res);
  }
//+------------------------------------------------------------------+
//| 返回格式化字符串表示方式                                           |
//+------------------------------------------------------------------+
string CRow::String(void) const
  {
   string out="";
//--- 如果数组大小大于零
   int size=ArraySize(m_array);
//--- 我们只使用一个非零的数组元素
   if(size>0)
     {
      out="{";
      for(int i=0;i<size;i++)
        {
         //--- 收集字符串的值
         out+=StringFormat(" %G;",m_array[i]);
        }
      out+=" }";
     }
//--- 结果
   return(out);
  }
//+------------------------------------------------------------------+
//| 类"Matrix"                                                       |
//+------------------------------------------------------------------+
class CMatrix
  {
private:
```

© 2000-2025, MetaQuotes Ltd.

```
   CRow              m_rows[];

public:
   //--- 构造函数和析构函数
                     CMatrix(void);
                     CMatrix(int rows)   { ArrayResize(m_rows,rows);             }
                    ~CMatrix(void){};
   //--- 得到矩阵大小
   int               Rows()        const { return(ArrayResize(m_rows));          }
   int               Cols()        const { return(Rows()>0? m_rows[0].Size():0); }
   //--- 以CRow行的形式返回列的值
   CRow              GetColumnAsRow(const int col_index) const;
   //--- 返回字符串矩阵值
   string            String(void) const;
   //--- 索引操作符返回一个字符串数字
   CRow *operator[](int i) const        { return(GetPointer(m_rows[i]));       }
   //--- 加法操作符
   CMatrix           operator+(const CMatrix &m);
   //--- 乘法操作符
   CMatrix           operator*(const CMatrix &m);
   //--- 赋值操作符
   CMatrix          *operator=(const CMatrix &m);
  };
//+------------------------------------------------------------------+
//| 默认构造函数,创建和零行的数组                                    |
//+------------------------------------------------------------------+
CMatrix::CMatrix(void)
  {
//--- 零矩阵的行数
   ArrayResize(m_rows,0);
//---
  }
//+------------------------------------------------------------------+
//| 返回CRow形式的列值                                               |
//+------------------------------------------------------------------+
CRow  CMatrix::GetColumnAsRow(const int col_index) const
  {
//--- 获得列值的变量
   CRow row();
//--- 矩阵的行数
   int rows=Rows();
//--- 如果大于零的行数,执行操作
   if(rows>0)
     {
      //--- 接收col_index索引列的值的数组
      double array[];
      ArrayResize(array,rows);
      //--- 填充数组
      for(int i=0;i<rows;i++)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 检查i行的列数 - 它可能超过数组的限制
          if(col_index>=this[i].Size())
            {
             Print(__FUNCSIG__,": Error! Column number ",col_index,"> row size ",i);
             break; // 行将是未初始化对象
            }
          array[i]=this[i][col_index];
         }
       //--- 基于数组值创建CRow行
       row=array;
      }
//--- 结果
   return(row);
  }
//+------------------------------------------------------------------+
//| 两个矩阵的加法                                                   |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator+(const CMatrix &m)
  {
//--- 通过矩阵的行数和列数
   int cols=m.Cols();
   int rows=m.Rows();
//--- 接收相加结果的矩阵
   CMatrix res(rows);
//--- 矩阵大小必须匹配
   if(cols!=Cols() || rows!=Rows())
     {
      //--- 无法相加
      Print(__FUNCSIG__,": Failed to add two matrices, their sizes are different");
      return(res);
     }
//--- 辅助数组
   double arr[];
   ArrayResize(arr,cols);
//--- 通过行相加
   for(int i=0;i<rows;i++)
     {
      //--- 写下数组中矩阵字符串的加法结果
      for(int k=0;k<cols;k++)
        {
         arr[k]=this[i][k]+m[i][k];
        }
      //--- 将数组放在矩阵行
      res[i]=arr;
     }
//--- 返回矩阵相加结果
   return(res);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 两个矩阵的乘法                                                   |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator*(const CMatrix &m)
  {
//--- 第一个矩阵的列数，通过矩阵的行数
   int cols1=Cols();
   int rows2=m.Rows();
   int rows1=Rows();
   int cols2=m.Cols();
//--- 接收相加结果的矩阵
   CMatrix res(rows1);
//--- 矩阵应该协调
   if(cols1!=rows2)
     {
      //--- 不能相乘
      Print(__FUNCSIG__,": Failed to multiply two matrices, format is not compatible "
            "- number of columns in the first factor should be equal to the number of
      return(res);
     }
//--- 辅助数组
   double arr[];
   ArrayResize(arr,cols1);
//--- 在矩阵乘法中填写该行
   for(int i=0;i<rows1;i++)// 通过行
     {
      //--- 重置接收数组
      ArrayInitialize(arr,0);
      //--- 通过行的元素
      for(int k=0;k<cols1;k++)
        {
         //--- 利用m矩阵的k列的值用于CRow
         CRow column=m.GetColumnAsRow(k);
         //--- 两行相乘，在i-th元素写下矢量标量乘法的结果
         arr[k]=this[i]*column;
        }
      //--- 将数组arr[]至于矩阵i-th行
      res[i]=arr;
     }
//--- 返回两个矩阵的产物
   return(res);
  }
//+------------------------------------------------------------------+
//| 赋值操作                                                         |
//+------------------------------------------------------------------+
CMatrix *CMatrix::operator=(const CMatrix &m)
  {
//--- 查找和设置行数
   int rows=m.Rows();
```

© 2000-2025, MetaQuotes Ltd.

```
   ArrayResize(m_rows,rows);
//--- 填充我们传递矩阵行值的行
   for(int i=0;i<rows;i++) this[i]=m[i];
//---
   return(GetPointer(this));
  }
//+------------------------------------------------------------------+
//| 矩阵的字符串表示                                                 |
//+------------------------------------------------------------------+
string CMatrix::String(void) const
  {
   string out="";
   int rows=Rows();
//--- 形成字符串的字符串
   for(int i=0;i<rows;i++)
     {
      out=out+this[i].String()+"\r\n";
     }
//--- 结果
   return(out);
  }
```

另见

重载, 算术运算, 函数重载, 优先规则

© 2000-2025, MetaQuotes Ltd.

# 外部函数描述

在其它模块中定义的外部函数必须明确描述。描述应包括返回类型，函数名称和输入参数序列以及它们的类型。这些描述缺失则会在编译、整编，或执行程序时导致错误。当描述一个外部对象时，使用关键字 *#import* 来指明模块。

例如：

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

在 import 的辅助下，很容易对即将从外部 DLL 或预编译的 EX5 程序库里调用的函数进行描述。EX5 程序库被编译为 ex5 文件，它具有 library (库文件) 属性。仅当函数以 export (导出) 修饰符描述后才可从 EX5 程序库里导入。

请留意，如果 DLL 和 EX5 程序库需要一同导入，则它们应具有不同的名称 (无论它们位于哪个目录里)。所有导入函数的解析域都与程序库的 "文件名"对应。

例如：

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

class CFoo
  {
public:
   int            GetLastError() { return(12345); }
   void           func()
     {
      Print(GetLastError());           // 调用类方法
      Print(::GetLastError());         // 调用 MQL5 函数
      Print(kernel32::GetLastError()); // 从 kernel32.dll 里调用 DLL 库函数
      Print(lib::GetLastError());      // 从 lib.ex5 里调用 EX5 库函数
     }
  };

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

参阅

© 2000-2025, MetaQuotes Ltd.

重载, 虚拟函数, 多态

© 2000-2025, MetaQuotes Ltd.

# 输出函数

mql5*输出*后置修饰符发布的函数可以用在其他mq5程序中。这种函数称为可输出函数，编辑后可以从其他程序调用。

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

该修饰符使编译器将函数添加进ex5文件导出的EX5函数表格中。只有带这个修饰符的函数可以从其他mq5程序中接入（"可见"）。

程序库属性通知编译程序EX5-文件是一个程序库，而编译程序会在EX5表头显示它。

所有计划输出的函数都要用*输出*修饰符标记。

另见

重载, 虚拟函数, 多态

© 2000-2025, MetaQuotes Ltd.

# 事件处理函数

MQL5语言提供预定义事件处理。处理这些事件的函数由MQL5程序确定；函数名，返回类型，参数组成（有几个参数情况下）以及类型必须严格遵照事件处理程序函数。

客户端的事件处理程序通过返回值类型和参数类型确定函数，来处理这个或者那个事件。如果其他参数，与下面描述不一致，但已被指定了相关的函数或者另一个返回类型，那么这个函数不能用于事件处理程序。

## OnStart

OnStart() 函数就是 启动 事件处理程序，运行脚本时自动生成。一定是空类型，无参数：

```
void OnStart();
```

对于OnStart()函数，指定整型 返回类型。

## OnInit

OnInit() 函数是 初始化 事件处理程序。必须是空型或者整型，无参数：

```
void OnInit();
```

初始化事件处理程序在EA交易或者指标下载后即时生成；它不生成脚本。OnInit()函数用于初始化。如果OnInit()返回值为整型，非零结果意味着初始化失败，生成初始化失败原因代码 REASON_INITFAILED 。

当返回INIT_FAILED时，EA会从图表中强制卸载。

返回INIT_FAILED时，指标不会从图表中卸载。图表上保留的指标无法运行 —事件处理程序不会在指标中调用。

要优化EA交易的输入参数，推荐使用ENUM_INIT_RETCODE枚举值作为返回代码。这些值用于组织优化的过程，包括选择最合适的测试代理。在EA交易的初始化期间测试开始之前您可以使用 TerminalInfoInteger() 函数请求一个代理的配置和资源信息(内核数量，空余内存额，等等)。根据所获得的信息，您可以在EA交易优化过程中选择允许使用这个测试代理，或拒绝使用它。

### ENUM_INIT_RETCODE

| 标识符 | 描述 |
| --- | --- |
| INIT_SUCCEEDED | 成功初始化，可以继续测试EA交易。<br>这段代码意味着与null值一样——在测试中EA交易已成功初始化。 |
| INIT_FAILED | 初始化失败；由于非常严重的错误而无法继续测试。例如，未能创建EA交易工作所需的指标。该返回值意味着等同非零值- 测试中初始化EA交易失败。测试给定的EA交易参数组将不会执行，代理可以任意接受新任务。 |
| INIT_PARAMETERS_INCORRECT | 这个值意味着错误的输入参数组。包含这个返回代码的结果字符串在普通优化表中以红色突出显示。 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| | 接到这个值后，该策略测试不会将这个任务确实地传递给其他代理进行重试。 |
| INIT_AGENT_NOT_SUITABLE | 在初始化期间没有错误，但出于某种原因，代理不适合测试。例如，没有足够的内存，没有OpenCL 支持，等等。返回这段代码后，代理将不会收到任务，直到这种优化结束。 |

空型OnInit()函数代表初始化成功。

## OnDeinit

OnDeinit() 函数称为失败初始化，是初始化失败事件处理程序。必须是空型且有一个包括初始化失败原因代码 的常量整型参数。如果声明不同类型，编译程序会发出警告，但函数不可调用。对于脚本来说不会生成初始化失败事件，因此OnDeinit()函数不用于脚本。

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

在以下情况下EA交易和指标生成初始化失败：Deinit事件在以下情况下为EA交易和指标而生成：

- 再次初始化前，mql5程序下的交易品种和图表周期发生变化；
- 再次初始化前，输入参数发生变化；
- mql5程序卸载前。

## OnTick

仅仅EA交易依附的图表中，交易品种收到新订单号时EA交易会生成 新订单号 事件。自定义指标或者脚本中确定OnTick()函数是无效的，因为订单号事件不为它们而生。

订单号事件只为EA交易而生，但是却不意味着EA交易需要OnTick()函数，因为EA交易不仅需要生成订单号，也需要生成计时器，预定事件和图表事件。这些都需要空型，无参数：

```
void OnTick();
```

## OnTimer

计时器事件发生时调用OnTimer()函数，由系统计时器生成，仅用于EA交易-不能用于脚本或者指标。函数EventSetTimer()接收事件，当同意该事件声明时，设置事件发生频率 。

可以用函数EventKillTimer()为特殊EA交易注销计时器事件。函数必须为空值，没有参数：

```
void OnTimer();
```

建议在OnInit() 函数中调用EventSetTimer() 函数，而EventKillTimer() 函数可以在OnDeinit()中调用。

每个EA交易，每个指标都有其独自的计时器且仅通过它来接收事件。如果创建了计时器但没有用EventKillTimer() 函数禁止，那么一旦mql5程序停止，计时器被强制破坏。

## OnTrade

© 2000-2025, MetaQuotes Ltd.

交易发生时调用这个函数，改变下订单和持仓列表，订单历史记录和交易历史记录时会出现。当交易活动执行挂单，持仓/平仓，停止设置，启动挂单等等，订单和交易历史记录或者仓位和当前订单列表也会相应改变。

```
void OnTrade();
```

接到这个事件时（需要交易策略状态）用户必须独自确认交易账户状态。如果函数OrderSend()成功调用，且返回了真实值，意味着交易服务器已经提交订单并确定了订单号。服务器一旦处理这个订单，交易事件就生成了。如果用户记住了订单号值，OnTrade() 函数执行时，用这个值可以看到订单情况。

## OnTradeTransaction

交易账户执行一些明确的行动时，其状态发生变化。这样的行为包括：

- 在客户端使用OrderSend和OrderSendAsync函数发送来自任何MQL5应用程序的交易请求及其进一步执行；
- 通过程序端图形界面发送交易请求和它的进一步执行；
- 挂单和停止订单在服务器上激活；
- 在交易服务器端执行操作。

这些操作的结果是会执行以下交易事务：

- 处理交易请求；
- 改变持仓订单；
- 改变订单历史；
- 改变交易历史；
- 改变持仓。

例如，当发送一个被处理的市场购买订单时，会为账户创建一个适当的购买订单，然后执行订单并从持仓列表中删除订单，然后将其添加到订单历史，适当的交易也会被添加到历史并创建一个新的持仓。所有这些行动都是交易事务。在程序端到这种事务就是一个TradeTransaction事件。它会调用OnTradeTransaction处理程序

```
void  OnTradeTransaction(
   const MqlTradeTransaction&    trans,        // 交易结构
   const MqlTradeRequest&        request,      // 请求结构
   const MqlTradeResult&         result        // 结果结构
   );
```

处理程序包含三个参数：

- **trans** – 该参数获得描述应用于交易账户的交易事务的 MqlTradeTransaction结构；
- **request** – 该参数获得描述交易请求的MqlTradeRequest结构；
- **result** – 该参数获得描述交易请求执行结果的MqlTradeResult结构。

最后两个**request**和**result**参数填充值仅用于TRADE_TRANSACTION_REQUEST类型事务，事务数据可以从 **trans**变量的 *type* 参数接收。注意在这种情况下，**trans**变量描述的交易事务被执行后，在**result**变量的*request_id*字段包含**request**交易请求的ID。请求ID允许联系执行的操作（OrderSend或OrderSendAsync函数调用）发送到OnTradeTransaction()该操作的结果。

© 2000-2025, MetaQuotes Ltd.

从程序端或通过OrderSend()/OrderSendAsync()函数手动发送的交易请求可以在交易服务器生成几个连续的事务。但不保证这些交易在程序端的优先到达。因此，开发您的交易算法时，您不必指望交易会一组接一组地到达。此外，交易可能在从服务器传递到终端过程中会丢失。

- 所有类型的交易事务都在ENUM_TRADE_TRANSACTION_TYPE枚举中描述。
- 交易事务中描述的MqlTradeTransaction结构根据事务类型以不同的方式填充。例如，只有类型字段（交易事务类型）必须为TRADE_TRANSACTION_REQUEST类型事务进行分析。OnTradeTransaction函数（请求和结果）的第二个和第三个参数必须为额外的数据进行分析。有关更多信息，请参见"交易事务结构"。
- 交易事务描述不提供关于订单，交易和持仓(如，评论)的所有可用的信息。OrderGet*, HistoryOrderGet*, HistoryDealGet*和PositionGet*函数应该用于获得扩展信息。

申请客户账户交易事务后，它们始终置于程序端交易事务队列，从这里它们一直送到OnTradeTransaction入口点以便到达程序端。

当使用OnTradeTransaction处理程序通过EA交易处理交易事务时，程序端会继续处理新来的交易事务。因此，交易账户的状态可以在OnTradeTransaction操作过程中已经改变。例如，当MQL5程序处理添加新订单的事件时，它可能被执行，从持仓列表删除并移动到历史记录。此外，应用程序将通知这些事件。

事务队列长度由1024个元素组成。如果OnTradeTransaction处理新事务太久，队列中的旧事务可能会被新的取代。

- 一般来说，没有准确的OnTrade和OnTradeTransaction调用数量的比例。一个OnTrade调用对应于一个或几个OnTradeTransaction调用。
- 适当地OnTradeTransaction调用后调用OnTrade。

## OnTester

函数OnTester()是外部EA交易历史测试结束后自动生成的测试事件处理程序。用双型确定函数，无参数：

```
double OnTester();
```

这个函数在OnDeinit() 之前调用，且有同样的双型返回值。只在EA交易测试时使用OnTester() 。其主要作用是计算某个值，该值用作遗传最优化自定义标准。

遗传优化中，生成结果用降序，例如从优化标准查看点，最佳结果是那些最大值（OnTester函数带进账户的最大自定义优化标准值）。这种情况下，最差值放在最后，或者排除，不参与下一步生成。

## OnTesterInit

当TesterInit事件发生时在EA中调用这个函数，以便在策略测试优化之前执行必要的操作。有两种函数类型。

返回结果的版本

```
int  OnTesterInit(void);
```

返回值

© 2000-2025, MetaQuotes Ltd.

int 类型值，0意味着优化开始之前图表上启动的EA的成功初始化。

建议使用返回执行结果的OnTesterInit()调用，因为它不仅可以程序初始化，还可以在早期优化停止的情况下返回一个错误代码。返回INIT_SUCCEEDED 以外的任何值，(0)表示错误，没有启动优化。

没有结果返回的版本只为与旧代码兼容而保留。不建议使用

```
void  OnTesterInit(void);
```

随着优化的开始，OnTesterDeinit()或OnTesterPass()处理程序的EA交易自动加载在一个测试中指定交易品种和周期的单独的程序端图表中，并接收TesterInit事件。优化启动前函数用于EA交易的初始化，用于进一步处理优化结果。

## OnTesterPass

OnTesterPass()函数是TesterPass事件的处理程序，这是在策略测试EA交易优化过程中收到帧时自动生成的。函数必须定义为void类型。它没有参数：

```
void OnTesterPass();
```

OnTesterPass()处理程序的EA交易自动加载在一个单独的指定交易品种/周期的程序端图表进行测试，并当在优化中接收到帧时获得TesterPass事件。函数用于“当场”动态处理优化结果无需等待其完成。使用FrameAdd()函数添加帧，该函数可以在OnTester()处理程序单次结束后被调用。

## OnTesterDeinit

OnTesterDeinit()是TesterDeinit事件的处理程序，它是策略测试EA交易优化结束后自动生成的。函数必须定义为void类型。它没有参数：

```
void OnTesterDeinit();
```

TesterDeinit()处理程序的EA交易在优化初期自动加载在图表上，并在其完成后接收TesterDeinit。该函数用于最终处理全部的优化结果。

## OnBookEvent

OnBookEvent()函数是BookEvent处理程序。 BookEvent只在市场深度改变时，转为EA交易而生成。它必须是空值，且有一个字符串类型参数：

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

若要接收任一交易品种的BookEvent，需要事先同意用函数MarketBookAdd()接收这个交易品种的BookEvent。如果不同接收某个特定交易品种的BookEvent，可以调用 MarketBookRelease()。

与其他事件不同的是BookEvent是直播。意味着如果EA交易同意用MarketBookAdd接收BookEvent，其他有OnBookEvent()处理程序的EA交易也会收到这个事件。因此分析交易品种名称是必须的，可以*const string& symbol*参数传到处理程序。

## OnChartEvent

OnChartEvent() 是ChartEvent的处理程序：

- CHARTEVENT_KEYDOWN —击键，图表窗口定格；

© 2000-2025, MetaQuotes Ltd.

- CHARTEVENT_MOUSE_MOVE — 鼠标移动事件和鼠标点击事件 (if CHART_EVENT_MOUSE_MOVE=true is set for the chart);
- CHARTEVENT_OBJECT_CREATE —创建图解物件 (if CHART_EVENT_OBJECT_CREATE=true is set for the chart);
- CHARTEVENT_OBJECT_CHANGE —通过属性对话框改变物件属性;
- CHARTEVENT_OBJECT_DELETE —删除图解物件 (if CHART_EVENT_OBJECT_DELETE=true is set for the chart);;
- CHARTEVENT_CLICK —鼠标单击图表;
- CHARTEVENT_OBJECT_CLICK —鼠标单击属于图表的图解物件;
- CHARTEVENT_OBJECT_DRAG —用鼠标移动图解物件;
- CHARTEVENT_OBJECT_ENDEDIT —图解物件标签编辑输入框中完成文本编辑;
- CHARTEVENT_CHART_CHANGE —图表事件变化;
- CHARTEVENT_CUSTOM+n —用户ID，0-65535之间范围。
- CHARTEVENT_CUSTOM_LAST —自定义(CHARTEVENT_CUSTOM +65535)接受的最后一个ID。

只在EA交易和指标中调用这个函数。函数是带有4个参数的空值：

```
void OnChartEvent(const int id,         // 事件 ID
                  const long& lparam,   // 长整型事件参量
                  const double& dparam, // 双精度事件参量
                  const string& sparam  // 字符串事件参量
  );
```

对于每一类型事件，函数OnChartEvent()输入参数都有处理事件的定值。通过参数传送的事件和值列在以下表格中。

| 事件 | id参数值 | lparam参数值 | dparam参数值 | sparam参数值 |
|---|---|---|---|---|
| 击键 | CHARTEVENT_KEYDOWN | 按键码 | 重复次数（用户控制按键后重复击键的次数） | 描述键盘按钮状态的位掩码的字符串值 |
| 鼠标事件（如果属性CHART_EVENT_MOUSE_MOVE=true图表设置） | CHARTEVENT_MOUSE_MOVE | X坐标 | Y坐标 | 描述鼠标按钮状态的位掩码的字符串值 |
| 创建图解物件（如果CHART_EVENT_OBJECT_CREATE=true图表设置） | CHARTEVENT_OBJECT_CREATE | — | — | 创建的图解物件的名称 |
| 通过属性对话框改变物件属性 | CHARTEVENT_OBJECT_CHANGE | — | — | 更改的图解物件名称 |
| 删除图解物件（如果 | CHARTEVENT_OBJECT_DELETE | — | — | 删除的图解物件名称 |

© 2000-2025, MetaQuotes Ltd.

| 事件 | id参数值 | lparam参数值 | dparam参数值 | sparam参数值 |
|---|---|---|---|---|
| CHART_EVENT_OBJECT_DELETE=true 图表设置) | | | | |
| 鼠标单击图表 | CHARTEVENT_CLICK | X坐标 | Y坐标 | — |
| 鼠标单击属于图表的图解物件 | CHARTEVENT_OBJECT_CLICK | X坐标 | Y坐标 | 事件发生的图解物件名称 |
| 用鼠标移动图解物件 | CHARTEVENT_OBJECT_DRAG | — | — | 移动的图解物件名称 |
| 图解物件标签编辑输入框中完成文本编辑 | CHARTEVENT_OBJECT_ENDEDIT | — | — | 文本编辑完成的图解物件名称 |
| 图表更改事件 | CHARTEVENT_CHART_CHANGE | — | — | — |
| N下的用户ID | CHARTEVENT_CUSTOM+N | 通过函数EventChartCustom() 设置值 | 通过函数EventChartCustom()设置值 | 通过函数EventChartCustom() 设置值 |

## OnCalculate

OnCalculate()函数只在自定义指标中调用，通过Calculate计算指标值是必须的。通常在接到指标计算的交易品种新订单号时发生。这个指标不需要附在交易品种的价格图表上。

OnCalculate() 函数有个返回值int。有两个可能定义。一个指标中不可以有两个函数版本。

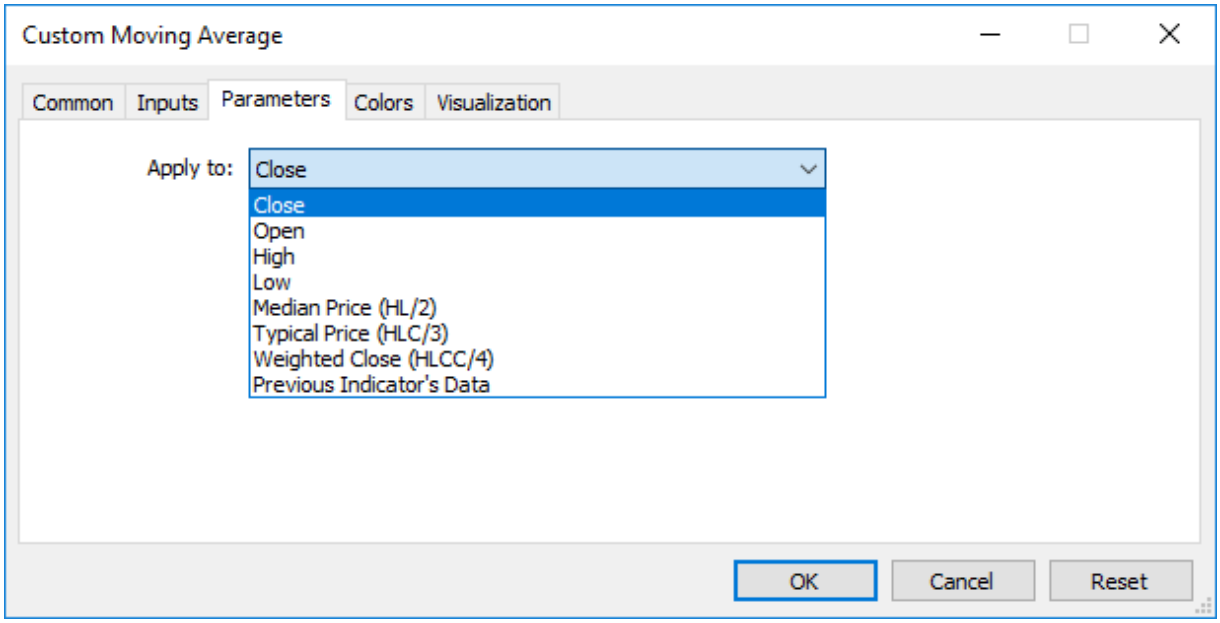
一种是用于单数据缓冲中计算的指标。例如，自定义移动平均数指标。

```
int OnCalculate (const int rates_total,      // 价格[] 数组的大小
                 const int prev_calculated,  // 前一次调用处理的柱
                 const int begin,            // 有效数据起始位置
                 const double& price[]       // 计算的数组
   );
```

价格[]数组中，可以传送时间序列和计算的一些指标缓冲。ArrayGetAsSeries()函数确定价格[]数组索引方向。为了不依赖默认值，需要无条件的调用ArraySetAsSeries()函数用于工作的数组。

价格[]数组中，在“参数”标签启动指标时，选择适当的时间序列或者指标。所以，需要在“应用于”字段的下拉列表中指定必要的项目。

© 2000-2025, MetaQuotes Ltd.




从其他mq5程序中接收自定义指标值，要使用iCustom()函数，返回嵌入指标处理程序。可以指定适当的价格□数组或者另一个指标处理程序。这个参数在自定义指标输入变量列表中最后传送。

示例：

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

这个示例中，通过的最后参数是PRICE_TYPICAL值（从ENUM_APPLIED_PRICE计数开始），指出自定义指标可以用获得的典型价格建立（高价+低价+平仓）/3。如果没有确定这个参数，指标基于PRICE_CLOSE值建立，例如每栏平仓价。

另一个示例显示依照指定价格□数组的最后一个参数传送指标处理程序，由函数iCustom()所描述。

另一种形式意在所有其他指标，计算更多的时间序列。

```
int OnCalculate (const int rates_total,      // 输入时间序列大小
                 const int prev_calculated,  // 前一次调用处理的柱
                 const datetime& time[],     // 时间
                 const double& open[],       // 开盘价
                 const double& high[],       // 最高价
                 const double& low[],        // 最低价
                 const double& close[],      // 收盘价
```

© 2000-2025, MetaQuotes Ltd.

```
                const long& tick_volume[],  // 订单交易量
                const long& volume[],       // 真实交易量
                const int& spread[]         // 点差
   );
```

开盘价[]，最高价[]，最低价[]和收盘价[]参数由当前时间表的开盘价，最高和最低价和收盘价数组组成。时间参数[]包括开盘时间值数组，扩展参数[]有一个数组包括扩展历史记录 如果为交易安全提供扩展)。volume[] 和tick_volume[] 参数分别包括交易和交易量历史记录。

确定时间[]索引方向， 开盘价[]， 最高价[]， 最低价[]， 收盘价[]， 交易量[]， 交易量[] 和 扩展[],需要调用ArrayGetAsSeries()函数。若不想依赖默认值，需要无条件的调用函数ArraySetAsSeries()用于工作的数组。

首先rates_total参数包括栏的数量，可用来计算指标，与图表中现存的栏数一致。

需要注意OnCalculate() 返回值和第二输入参数prev_calculated的连接。调用函数时，prev_calculated 参数包括上次调用时OnCalculate() 返回值。这就允许用经济算法计算自定义指标，避免重复计算。

返回rates_total参数值足够了，包括当前调用函数的栏数。如果自从上次调用函数OnCalculate()，价格数据更改了（下载深度历史记录或者填满历史空白期），输入参数prev_calculated 值由终端机设置为零。

注：如果OnCalculate返回零，那么指标值不能显示在客户端的数据窗口。

为更好的理解，启动附加以下代码的指标很有用。

指标示例：

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图的线
#property indicator_label1  "Line"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDarkBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标缓冲区
double         LineBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime& time[],
                const double& open[],
                const double& high[],
                const double& low[],
                const double& close[],
                const long& tick_volume[],
                const long& volume[],
                const int& spread[])
  {
//--- 获得当前交易品种和图表周期的有效柱数
   int bars=Bars(Symbol(),0);
   Print("Bars = ",bars,", rates_total = ",rates_total,",  prev_calculated = ",prev_ca
   Print("time[0] = ",time[0]," time[rates_total-1] = ",time[rates_total-1]);
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

另见

运行程序，客户端事件，工作事件

© 2000-2025, MetaQuotes Ltd.

# 变量

## 变量声明

可变量必须在声明之前使用。可变量必须拥有特殊的辨认名。相关可变量的定义描述会显示。

基本类型如下：

- 字符型，短整型，整型，长整型，无符号字符型，无符号短整型，无字符整型，无符号长整型-整数；
- 颜色-代表RGB-颜色的整数；
- 日期时间-日期和时间，自1970年1月1日起无符号整数包括秒数；
- 布尔数据-布尔值的 *true* 和 *false*；
- 双精度数字-带有浮点的双精度数字；
- 浮点型-带有浮点的单精度数字；
- 字串符数据-特殊字符串。

示例：

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

复合类型：

用其他类型构成的数据类型结构。

```
struct MyTime
  {
   int hour;     // 0-23
   int minute;   // 0-59
   int second;   // 0-59
  };
...
MyTime strTime; // 预先声明的结构MyTime MyTime的变量
```

结构类型声明前不可结构类型变量声明。

数组

相同数列数据被标注序列：

```
int    a[50];         // 50个整数的一维数组
double m[7][50];      // 7个数组的二维数组，
                      // 每个都由50个数字组成。
MyTime t[100];        // 数组包含元素例如 MyTime
```

© 2000-2025, MetaQuotes Ltd.

唯一整数可以是数组指数。不允许四唯数列。数组元素开始编号为0。一个一维列阵的最后元素是1的数字比列阵大小。这就意味着，请求数列的最后元素包括50个整数将出现作为a[49]。维度被标注从0到维度大小-1. 一个二维数组的最后元素从示例将出现作为m[6][49]。

静态数组不能代表时间序列，例如ArraySetAsSeries()函数从后到前接入数组，不能应用于此。如果想要在时间序列中接入数组，使用动态数组对象。

如果访问超出数列范围，执行的子系统将生成严重错误，程序就会停止。

## 用于处理数组的内置方法

来自数组函数部分的函数，以及内置方法可以用来处理数组：

| 方法 | Analog | 描述 |
|---|---|---|
| void array.Fill(const scalar value, const int start_pos=0, const int count=-1); | ArrayFill, ArrayInitialize | 用指定的值填写数组 |
| void array.Free(); | ArrayFree | 释放动态数组缓存区并将零维度大小设置为0（零） |
| int array.Resize(const int range0_size, const int reserve); int array.Resize(const int range_sizes[], const int reserve); | ArrayResize | 在数组第一维度中设置新大小 |
| int array.Print(); | ArrayPrint | 在日志中显示简单的类型数组值 |
| int array.Size(const int range=-1); | ArraySize, ArrayRange | 返回整个数组（范围=1）或指定数组维度中的元素个数 |
| bool array.IsDynamic(); | ArrayIsDynamic | 检查数组是否为动态 |
| bool array.IsIndicatorBuffer(); | | 检查数组是否为指标缓存区 |
| bool array.IsSeries(); | ArrayIsSeries | 检查数组是否为时间序列 |
| bool array.AsSeries(); | ArrayGetAsSeries | 检查数组索引方向 |
| bool array.AsSeries(const bool as_series); | ArraySetAsSeries | 设置数组中的索引方向 |
| int array.Copy(const src_array[], const int dst_start, const int src_start, const int cnt); | ArrayCopy | 将数组值复制到另一个数组 |
| int array.Compare(const src_array[], const int dst_start, const int src_start, const int cnt); | ArrayCompare | 返回比较两个简单类型数组或自定义结构的结果 |
| int array.Insert(const src_array[], const int dst_start, const int src_start, const int cnt); | ArrayInsert | 从指定索引开始，将来自源数组的指定元素数量插入到接收数组 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | Analog | 描述 |
| --- | --- | --- |
| int array.Remove(const int start_pos, const int count); | ArrayRemove | 从指定索引开始移除数组中指定的元素数量 |
| int array.Reverse(const int start_pos, const int count); | ArrayReverse | 从指定索引开始反转数组中指定的元素数量 |
| bool array.Swap(array& arr[]); | ArraySwap | 与另一个相同类型的动态数组交换内容 |
| void array.Sort(sort_function); | ArraySort | 按第一个维度对数值数组进行排序 |
| int array.Search(scalar value, search_function); | ArrayBsearch | 返回在数组第一维度中检测到的第一个元素的索引 |
| int array.Find((scalar value, search_function); | | 使用传递的函数在数组中执行搜索并返回第一个检测到的元素的索引 |
| array array.Select(scalar value, search_function); | | 使用传递的函数在数组中执行搜索并返回包含所有检测到的元素的数组 |

## 接入说明符

接入说明符表示编译器如何接入变量，结构会员或者类。

常量 说明符声明常数变量，运行期不允许改变这个变量。变量声明时允许独自初始化。

样本

```
int OnCalculate (const int rates_total,      // 价格[] 数组的大小
                 const int prev_calculated,  // 前一次调用处理的柱
                 const int begin,            // 有效数据起始位置
                 const double& price[]       // 数组计算
   );
```

若要接入结构会员和类使用以下限定符：

- 公用限定符 -允许自由接入变量或者类函数
- 受保护限定符 -允许从这个类函数或者公共继承类函数中接入。不允许其他接入；
- 私人限定符-允许仅从同类函数中接入变量和类函数。
- 虚拟限定符 -仅应用于类函数（不是结构函数）并且告诉编译器这个函数置于类虚拟函数表格中。

## 存储类

有三种存储类: 静态, 输入, 外部。存储类的这些调节器明确表明内存预分配区分配的相关变量的编译器，称为全局池。此外，这些调节器表示特别处理变量数据。如果局部声明的变量不是

© 2000-2025, MetaQuotes Ltd.

静态，这个变量内存自动分配程序堆。非静态数组分配的自由内存也是在超出声明数组可见区时自动完成。

另见

数据类型，类型密封和扩展，变量初始化，可见范围和变量使用期，创建和删除对象，Static Members of a Class

© 2000-2025, MetaQuotes Ltd.

# 局部变量

在函数内可变量的声明是局部的。局部变量在声明的部分里是被限定的。局部变量可以由任意一个表达式结果初始化。 每次函数的运行只可以初始化一个局部变量。局部变量储存在相应的存储器上。

示例：

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

变量范围是涉及变量的模块部分。模块中声明的变量（内部），有模块规定其范围。变量声明启动模块范围，以终止程序结束。

开始函数声明的局部变量也有模块范围，和局部变量函数参数。任何模块都可以包括变量声明。如果嵌入模块内外模块有同名标识符，隐藏外部指标，直到内部操作完成。

示例：

```
void OnStart()
  {
//---
   int i=5;       // 函数的局部变量
     {
      int i=10;  // 函数变量
      Print("In block i = ",i); // 结果是 i=10;
     }
   Print("Outside block i = ",i);  // 结果是 i=5;
  }
```

这意味着内部模块运行时，只看到自己的局部指标值，看不到外部同名指标值。

示例：

```
void OnStart()
  {
//---
   int i=5;       // 函数局部变量
   for(int i=0;i<3;i++)
      Print("Inside for i = ",i);
   Print("Outside the block i = ",i);
  }
/* Execution result
Inside for i = 0
Inside for i = 1
Inside for i = 2
Outside block i = 5
*/
```

© 2000-2025, MetaQuotes Ltd.

即使从启动模块就存在，公开静态局部变量仍有模块范围。

## 存储栈

在每个MQL5程序中，称为存储栈的特殊内存区被分配存储自动创建的局部函数变量。一个存储栈可以分配给全部函数，指标的默认大小等于1Mb。在EA交易和脚本中，存储栈大小都可以使用#property stacksize编译指令（以字节设定存储栈大小）来管理，默认情况下，为存储栈分配8Mb内存。

静态局部变量存储在与其他静态和全局变量存储的相同的地方 - 特殊的内存区域，它是独立于存储栈的存在。动态创建的变量也使用独立于存储栈的内存区域。

随着每个函数的调用，存储栈上的区域分配给内部非静态变量。退出函数后，内存可再次提供使用。

如果从第一个函数调用第二个，然后第二个函数从剩下的栈内存占据变量所需的大小。因此，当使用包含函数时，每个函数将会连续占据栈内存。这可能会导致一个函数调用期间的内存不足，这种情况被称为存储栈溢出。

因此，对于大型本地数据，您应该更好地利用动态内存 - 当进入一个函数时，分配本地所需的内存，在系统（new, ArrayResize()），和退出函数时，释放内存（delete, ArrayFree()）。

另见

数据类型, 类型密封和扩展, 变量初始化, 可见范围和变量使用期, 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 形式参数

通过函数的变量是局部的。范围是在作用块内。在作用之内正式变量的名称必须不同于其他外部定义变量和函数变量。作用块内的正式变量值已经被赋予。如果正式参数以常量修饰符声明，其值在函数中不能改变。

示例：

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

正式参量可能由常数初始化。在这种情况下，初始化的值作为缺省值被考虑。参量，在初始化旁边，必须初始化。

示例：

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

这样作用显现时，初始化的参量可能被省去，缺省值会代替它们。

示例：

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

简易类型参量值通过。在任何情况下，这种局部变量的修改将不会显示在功能板块内。任何数组和结构类型数据都是通过引用传送。如果需要禁止修改数组和结构内容，这些类型参数必须用关键字 *const* 声明。

有机会通过引用传送简单类型参数。在这种情况下，修改的这样参量将影响调用函数对应的变量。为指出引用传送参数，数据类型后放符号&。

示例：

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

© 2000-2025, MetaQuotes Ltd.

引用传送的参量无法通过默认值初始化。

传到函数的最大参量不可以超过64个。

另见

输入变量，数据类型，类型密封和扩展，变量初始化，可见范围和变量使用期，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 静态变量

静止存储类称为静态变量。在数据类型之前指定静态修饰符。

示例：

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

与简单的仅能用表达式初始化的局部变量不同的是，静态变量可以通过相关的常量和常量表达式初始化。

静态变量存在于程序执行期，在特别函数OnInit()调用前初始化。如果未指定初始化值，静态变量为零初始化值。

关键字 *static* 声明的局部变量通过函数使用期保留值。下次调用函数时，局部变量会包含上次调用的值。

任何变量，除了函数的形式参数，都称为静态变量。如果局部变量声明不是静态，变量内存会在程序中自动分配。

示例：

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

另见

数据类型，类型密封和扩展，变量初始化，可见范围和变量使用期，创建和删除对象，Static Class Members

© 2000-2025, MetaQuotes Ltd.

# 全局变量

全局变量通过函数外部描述声明而创建。全局变量作为函数被定义在相同水平，即，不可以局部使用。

示例：

```
int GlobalFlag=10;   // 全局变量
int OnStart()
  {
   ...
  }
```

全局变量的范围是整个程序。全局变量在所有程序内是被定义的。如果它的值没有被定义，初始化值为零。全局变量只对于相应的常数初始化。

全局变量只在程序载入客户端内存之后和初次处理Init事件之前初始化一次。对于代表类对象的全局变量，初始化期间调用相应的构造函数。脚本中，全局变量在处理Start事件之前进行初始化。

注解：变量在全局变量的水平位上不能够与客户端GlobalVariable...() 函数混淆。

另见

数据类型, 类型密封和扩展, 变量初始化, 可见范围和变量使用期, 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 输入变量

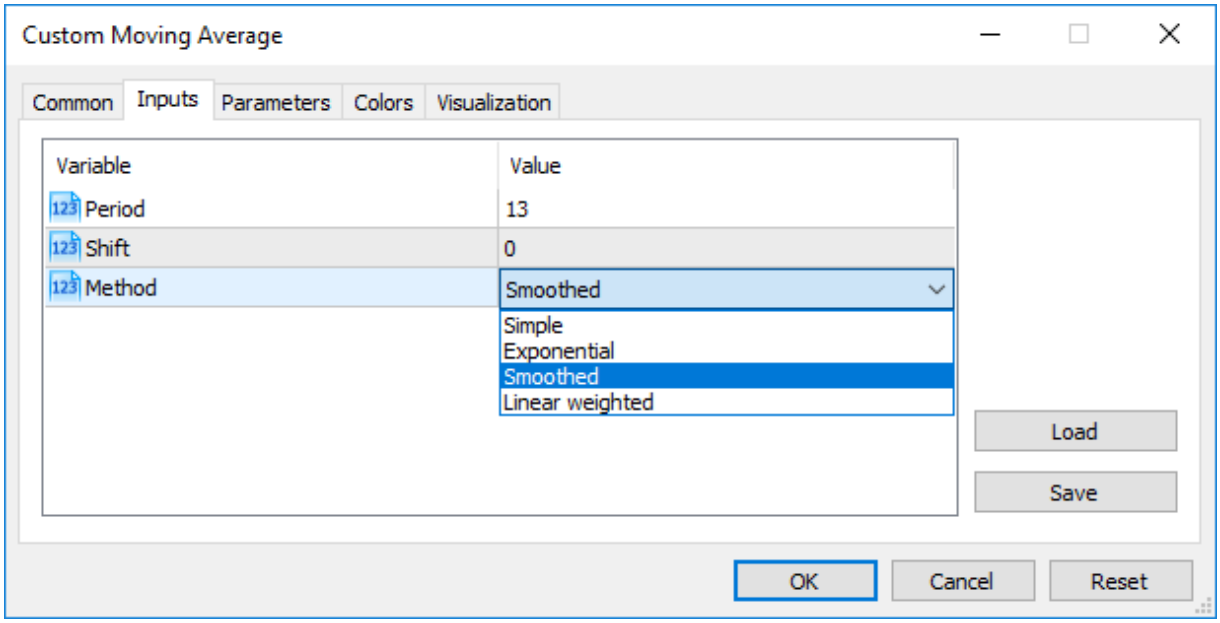
输入存储类称作外部变量。*input*修饰符位于数据类型前。输入修饰符变量不可以在mql5程序中修改，只能只读访问这个变量。用户只能通过程序属性窗口改变输入变量值。

输入变量名的最大长度为63个字符。对于string类型的输入参数，最大值长度（字符串长度）可以从191到253字符（参见注意）。最小长度为0字符（该值未设置）。

示例：

```
//--- 输入参量
input int             MA_Period=13;
input int             MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMMA;
```

输入变量决定程序的输入参数。可以在程序属性窗口中得到。

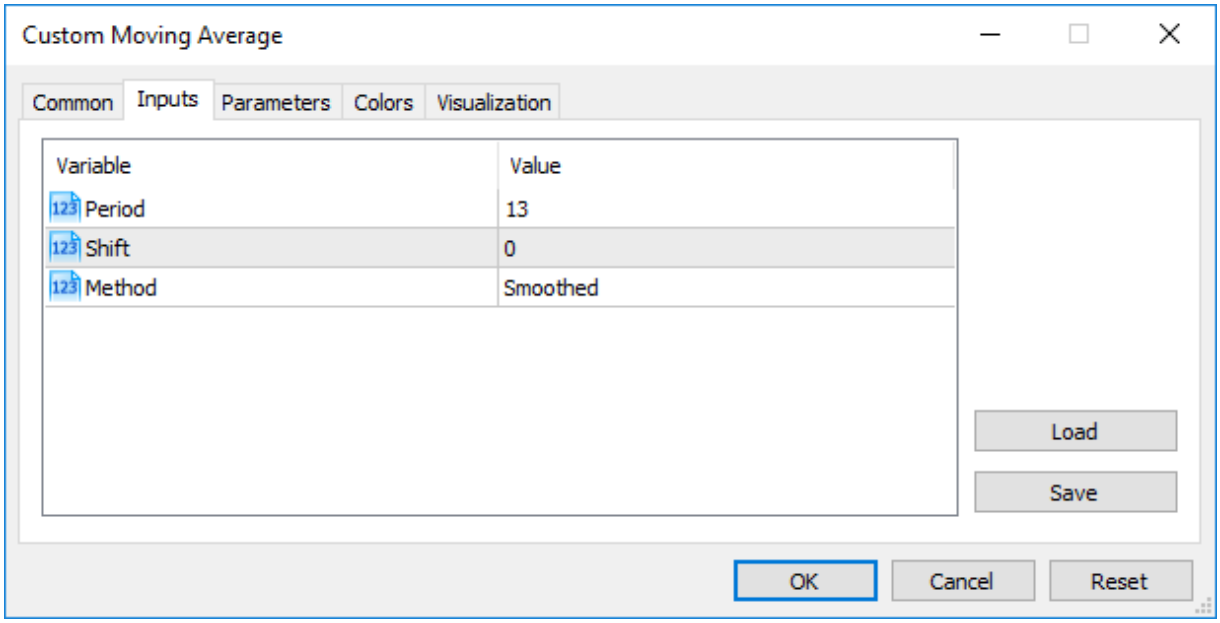



输入标签中可以通过其他方法显示输入参数名。使用字符注解，放在同行输入参数描述后。因此对于用户来说，与名字匹配的参数更容易理解。

示例：

```
//--- 输入参量
input int             InpMAPeriod=13;          // 平滑周期
input int             InpMAShift=0;            // 水平移动行
input ENUM_MA_METHOD InpMAMethod=MODE_SMMA;  // 平滑函数
```

© 2000-2025, MetaQuotes Ltd.




注意：复合型数组和变量不能作为输入变量。

注意：输入变量的字符串注释长度不得超过63个字符。

注意：对于string类型的输入变量，值长度（字符串长度）的限制由以下条件设置：

- 参数值由"parameter_name=parameter_value"字符串表示（考虑'='），
- 最大表示长度为255个字符（*total_length_max*=255或254个字符，不包括'='），
- *parameter_name_length*字符串参数的最大长度 = 63个字符。

因此，字符串参数的最大字符串大小使用公式计算：

```
parameter_value_length=total_length_max-parameter_name_length=254-parameter_name_lengt
```

这里最大字符串大小从191（parameter_name_length=63）到253个字符（parameter_name_length=1）。

## MQL5程序访问自定义指标时传送参数

用iCustom()函数访问自定义指标。其名后，参数要与该指标声明的输入变量严格一致。如果参数少于变量，缺失的参数要在变量声明时用特殊值填满。

如果自定义指标使用第一型的OnCalculate函数（例如用同数据数组计算的指标），那么ENUM_APPLIED_PRICE值或者其他指标处理程序作为访问这个自定义指标的上一个参数被使用。与输入变量一致的所有参赛必须清楚指明。

## 计数作为输入参数

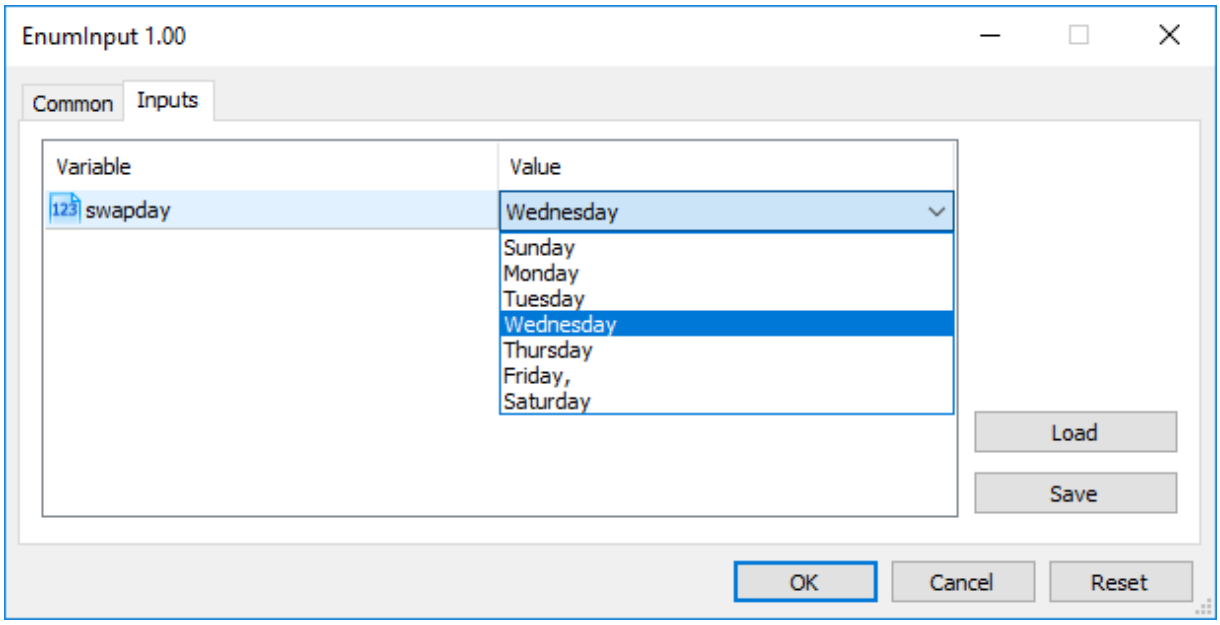
MQL5提供得嵌入计数和用户指定的变量都可以用作输入变量（mql5程序的输入参数）。例如，可以创建dayOfWeek计数，描述一周的天数，使用输入变量指定某一天，不是数字，用更通常的方法。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property script_show_inputs
//--- 一周内每天
enum dayOfWeek
  {
   S=0,      // 周日
   M=1,      // 周一
   T=2,      // 周二
   W=3,      // 周三
   Th=4,     // 周四
   Fr=5,     // 周五,
   St=6,     // 周六
  };
//--- 输入参量
input dayOfWeek swapday=W;
```

脚本启动时，为使用户从属性窗口选择必要的值，使用预处理命令#property script_show_inputs。启动脚本并从列表中选择一个dayOfWeek列举值。启动Enum InInput脚本然后去输入标签。默认情况下，swapday值（三个交换日）是周三（W=3），但可以指定另一个值，用这个值改变程序操作。




列举值有限。因此，若要选择输入值，需使用下拉列表。为列表中显示的值使用列举助记名。如果注释和助记名有关，正如示例显示，注释内容代替助记名。

每个dayOfWeek计数值都是0-6，但是参数列表中，显示每个值的注释。明确描述输入参数使编程更灵活。

## Sinput变量修饰符

输入变量修饰符不仅允许启动程序时设置外部参数值也是策略测试中优化交易策略时所必须的。每个不包括字符串类型的输入变量可用于优化。

© 2000-2025, MetaQuotes Ltd.

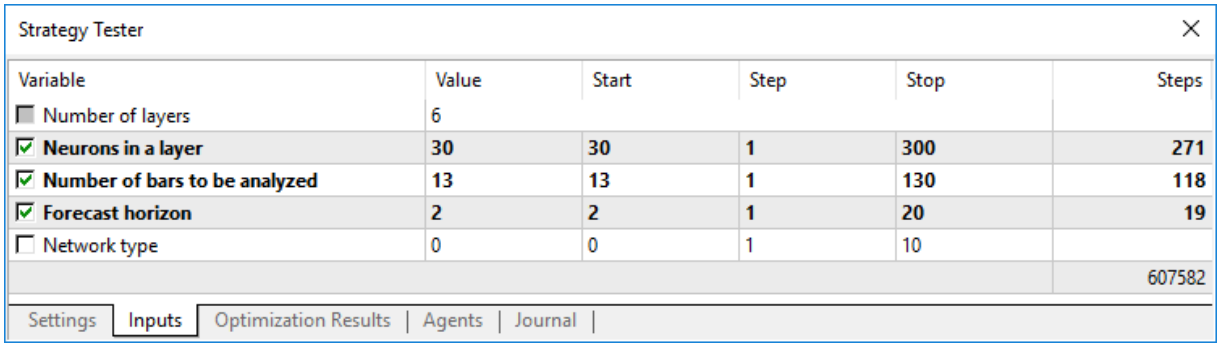
有时，需要从测试的所有通过的区域排除一些外部程序参数。这种情况下，就会引入sinput内存修饰符。sinput代表静态外部变量声明(sinput=静态输入)。这意味着EA交易代码的以下声明。

```
sinput       int layers=6;   // 层数
```

将相当于全面声明

```
static input int layers=6;   // 层数
```

Sinput修饰符声明的变量是MQL5程序的一个输入参数。当启动程序时可以更改该参数的值。然而，这个变量不用于输入参数的优化。换句话说，当寻找适合指定条件的最好的一组参数时不会枚举它的值。




上面所示的EA交易有5个外部参数。"层数"声明输入并等于6。交易策略优化期间该参数不能改变。我们可以指定必要的值用于进一步使用。开始，步骤和停止字段都不可用于这样的一个变量。

因此，为变量指定sinput修饰符之后用户将无法优化该参数。换句话说，程序端用户将无法在策略测试为其设置初始和最终值用于优化期间指定范围的自动枚举。

然而，该规则有一个例外：sinput变量可以使用ParameterSetRange()函数在优化任务中有所不同。这个函数的引入是专门用于任何输入变量可用值集的程序控制，包括声明为静态输入(sinput)的变量。ParameterGetInput()函数允许接收优化启动(在OnTesterInit()处理程序)时的输入变量值和重置步骤变化值和范围，在这范围内将会枚举优化参数值。

通过这种方式，结合sinput修饰符和输入参数使用的两个函数，根据另一个输入参数值，允许创建一个灵活的设置输入参数优化间隔的规则，。

## 排列输入参数

为了便于使用MQL5程序，可以使用group关键字将输入参数划分为命名块。这可以根据嵌入其中的逻辑将一些一些参数与其他参数进行可视化分离。

```
input group           "Group name"
input int             variable1 = ...
input double          variable2 = ...
input double          variable3= ...
```

这样声明之后，当在图表或策略测试中启动时，所有输入参数都被可视化地连接到一个指定的组中，从而简化MQL5用户的参数配置。每个组的规格在一个新组声明出现之前都一直有效：

© 2000-2025, MetaQuotes Ltd.

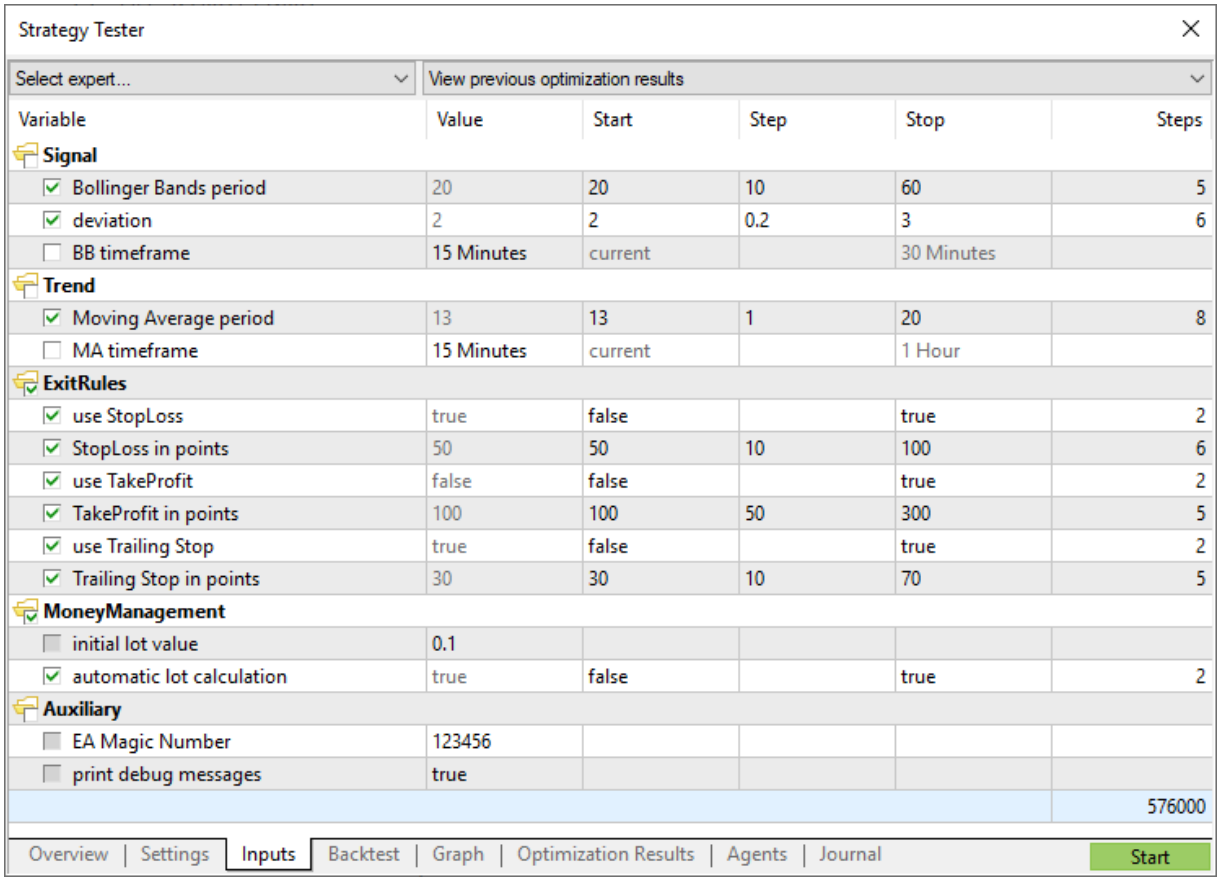
```
input group           "Group name #1"
input int             group1_var1 = ...
input double          group1_var2 = ...
input double          group1_var3 = ...

input group           "Group name #2
input int             group2_var1 = ...
input double          group2_var2 = ...
input double          group2_var3 = ...
```

样例EA包含根据用途划分的输入块：

```
input group           "Signal"
input int             ExtBBPeriod    = 20;        // 布林带周期
input double          ExtBBDeviation= 2.0;        // 偏差
input ENUM_TIMEFRAMES ExtSignalTF=PERIOD_M15;    // BB 时间周期

input group           "Trend"
input int             ExtMAPeriod    = 13;        // 移动平均线周期
input ENUM_TIMEFRAMES ExtTrendTF=PERIOD_M15;     // MA 时间周期

input group           "ExitRules"
input bool            ExtUseSL       = true;      // 使用“止损”
input int             Ext_SL_Points = 50;         //  “止损”点数
input bool            ExtUseTP       = false;     // 使用“止盈”
input int             Ext_TP_Points = 100;        //  “止盈”点数
input bool            ExtUseTS       = true;      // 使用“追踪止损”
input int             Ext_TS_Points = 30;         //  “追踪止损”点数

input group           "MoneyManagement"
sinput double         ExtInitialLot = 0.1;        // 初始手数值
input bool            ExtUseAutoLot = true;       // 自动手数计算

input group           "Auxiliary"
sinput int            ExtMagicNumber = 123456;    // EA幻数
sinput bool           ExtDebugMessage= true;      // 打印调试信息
```

当在策略测试中启动这种EA时，您可以双击组名来折叠/扩展输入参数块，也可以点击组复选框来选择其所有参数进行优化。

© 2000-2025, MetaQuotes Ltd.




另见

iCustom, 枚举, 程序属性

© 2000-2025, MetaQuotes Ltd.

# 外部变量

关键字extern 用来声明变量标识符，作为全局使用期的静态存储标识符。从程序启动，这些变量即存在，其内存会在程序启动后即时分配和初始化。

创建多源文件程序；这个情况下直接使用预处理#include 。外部声明的同一类型和标识符的变量位于一个项目中的不同源文件。

编译整个项目时，所有同型同标识符的外部变量与全局变量池相连。外部变量用来分离编译源文件。外部变量可以初始化，只能一次-禁止几个同型同标识符外部变量初始化。

另见

数据类型, 类型密封和扩展, 变量初始化, 可见范围和变量使用期, 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 变量初始化

下定义时可以初始化任何变量。如果变量没有显性初始化，其值可为任何值。不可使用隐性初始化。

全局和静态变量仅可以通过相关的常量或者常量表达式初始化。全局变量仅可以通过表达式初始化（不是常量）。

全局变量和静态变量初始化仅可以执行一次。局部变量每次调用相关函数都进行初始化。

示例：

```
int    n       = 1;
string s       = "hello";
double f[]     = { 0.0, 0.236, 0.382, 0.5, 0.618, 1.0 };
int    a[4][4] = { {1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {4, 4, 4, 4 } };
//--- 来自俄罗斯方块
int    right[4]={WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER,
                 WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER};
//--- 零值结构所有字段的初始化
MqlTradeRequest request={};
```

数组元素值列表必须被附寄在括号内。初始化省去的值被考虑为零。

如果没有指定初始化数组大小，则由基于初始化序列大小的编译器决定。

示例：

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

© 2000-2025, MetaQuotes Ltd.

对于结构类型，允许变量分批的初始化，静态数组(隐式地设置大小)也一样。您可以初始化一个或多个结构或数组的最初元素，在这种情况下，其他元素将被零初始化。

另见

数据类型, 类型密封和扩展, 可见范围和变量使用期, 创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 可见范围和变量使用期

有两种基本范围：局部范围和全局范围。

所有函数外部变量声明位于全局范围。 程序的任何地方都可以接入这个变量。这个变量位于内存全局池，所以其使用期同于程序使用期。

内部变量声明属于局部范围。这种变量外部不可见 因此无法得到）。大部分局部变量声明都是一个函数内声明的。局部变量声明位于存储器中，使用期同于函数使用期。

局部变量范围在存储器中声明，也可以在其他存储器以同名声明；也可以高级别声明，乃至全局范围。

示例：

```
void CalculateLWMA(int rates_total,int prev_calculated,int begin,const double &price[]
  {
   int        i,limit;
   static int weightsum=0;
   double     sum=0;
//---
   if(prev_calculated==0)
     {
      limit=MA_Period+begin;
      //--- 为第一限价柱设置空值
      for(i=0; i<limit; i++) LineBuffer[i]=0.0;
      //--- 首先计算明显值
      double firstValue=0;
      for(int i=begin; i<limit; i++)
        {
         int k=i-begin+1;
         weightsum+=k;
         firstValue+=k*price[i];
        }
      firstValue/=(double)weightsum;
      LineBuffer[limit-1]=firstValue;
     }
   else
     {
      limit=prev_calculated-1;
     }

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

© 2000-2025, MetaQuotes Ltd.

注意变量i，直线声明

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

其范围只是一个循环；外圈是同名的函数开始就声明的另一个变量。此外，变量k在内部声明，其范围就是循环内部。

访问静态说明符局部变量声明。这种情况下，编译器在内存全局池有一个变量。因此，静态变量使用期等于程序使用期。这就是变量范围的限制。

另见

数据类型，类型密封和扩展，变量初始化，创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 创建和删除对象

mql5程序执行后，内存会按照变量类型分配。根据接入水平，所有变量分成两类-全局变量和局部变量。根据内存等级，它们可以是mql5的输入参数，静态的或者自动的。如果需要，每个变量用相应值初始化。使用后，程序结束，使用的内存回到MQL5执行系统中。

## 初始化和无法初始化全局变量

mql5程序载入以及调用函数前，全局变量自动初始化。初始化过程是，初始值分配到简易类型变量中，并且调用构造函数（若有的话）。全局范围声明输入变量，程序启动时在对话框中设置初始值。

即使通常在局部范围声明静态变量，但是与全局变量一样，仍要预分配内存，程序载入后初始化。

初始化命令与程序中声明的变量命令一致。相反就执行无法初始化命令。这个规则只适用于非新操作符创建的变量。载入后，变量自动创建和初始化，程序卸载前，无法初始化。

## 初始化和无法初始化局部变量

如果声明局部变量不是静态的，内存会自动分配。局部变量与全局变量一样，程序遇到局部变量时自动初始化。因此初始化命令与声明命令一致。

局部变量在程序结束时无法初始化。程序模块是复合操作符，是选择操作符切换，循环操作符（for, while, do-while），函数主体，或者if-else操作符的一部分，

局部变量只在执行程序遇到变量声明时初始化。如果程序执行时，分程序没有执行，那么变量不会初始化。

## 安置对象的初始化和无法初始化

特殊情况就是对象指针 ，因为声明指针无能导致相关对象初始化。动态的安置对象仅当新操作符创建简单等级时初始化。对象初始化假定调用相关类的构造函数。如果没有类的相关构造函数，简易类型不会自动初始化；字符串，动态数组 和 复合对象 也会自动初始化。

指针在局部或者全局声明；可以通过空值NULL或者同指针值或遗传类型值初始化。如果*新*操作符为局部声明指针调用，这个指针的*删除*操作符退出前也要执行。否则指针消失，对象显性删除失败。

通过表达式*object_pointer*=new *Class_name*创建的所有对象一定通过删除操作符(*object_pointer*)来删除。如果程序完成后因为某种原因无法用删除 操作符删除变量，“专家”日志中会出现相应输入。可以声明几个变量，然后给它们安排一个指针。

如果动态创建的对象有构造函数，执行*新*操作符时调用这个函数。如果对象有析构函数，执行*删除*操作符是使用。

因此动态安置的对象只有用*新*操作符创建，而用*删除*操作符删除或者执行MQL5卸载程序时删除。动态创建的对象声明的指针不会影响其初始化命令。初始化和无法初始化由程序员控制。

## MQL5的动态存储分配

© 2000-2025, MetaQuotes Ltd.

当使用动态数组时，释放的内存立即返回到操作系统。

当使用new操作符从事动态类对象工作时，会从内存管理器工作的类内存池请求最初的内存。如果内存池中没有足够的内存，则从操作系统请求内存。当使用delete 操作符 删除动态对象时，释放的内存立即返回给类内存池。

内存管理器退出下面事件处理函数后立即释放内存返回操作系统：OnInit(), OnDeinit(), OnStart(), OnTick(), OnCalculate(), OnTimer(), OnTrade(), OnTester(), OnTesterInit(), OnTesterPass(), OnTesterDeinit(), OnChartEvent(), OnBookEvent()。

## 变量简要特征

创建，删除，调用构造函数和析构函数命令的信息如下表。

| | 全局自动变量 | 局部自动变量 | 动态创建对象 |
|---|---|---|---|
| 初始化 | 加载mql5后 | 执行中公开代码线到达时 | 执行新操作符 |
| 初始化命令 | 初始化命令 | 命令声明 | 不考虑命令声明 |
| 无法初始化 | mql5程序卸载前 | 推出公示分程序 | 执行删除操作符或者mql5卸载前 |
| 无法初始化命令 | 与初始化命令相反的命令 | 与初始化命令相反的命令 | 不考虑初始化命令 |
| 调用构造函数 | 加载mql5程序 | 初始化 | 执行*新*操作符 |
| 调用析构函数 | 卸载mql5程序 | 退出变量初始化分程序 | 执行*删除*操作符 |
| 错误日志 | “专家”日志中尝试删除自动创建的对象 | “专家”日志中尝试删除自动创建的对象 | “专家”日志，卸载mql5程序时不删除动态穿件的对象。 |

另见

数据类型, 类型密封和扩展, 变量初始化, 可见范围和变量使用期

© 2000-2025, MetaQuotes Ltd.

# 预处理程序

预处理程序是一个特殊MQL4的子程序，在程序执行之前预先准备的程序源代码。

预处理程序会尽可能地读取源代码。代码的结构可能包括MQL4 程序源代码的特殊文件。 对于读取的代码尽可能地按照具体常数分配储存。

预处理程序允许mql5 程序参量指定。

- 常量声明
- 设置程序属性
- 程序文本文件
- 输入函数
- 条件编译

编译器使用预处理程序指令在编译之前对源代码进行预处理。这个指令通常以#开始，因此编译器禁止在变量，函数等名称中使用符号。

每个指令都由一个单独的条目描述，并且在换行符之前都是有效的。在一个条目中不能使用多个指令。如果指令条目太大，可以使用'\'符号将其分隔成若干行。 在这种情况下，下一行被认为是指令条目的延续。

```
//+------------------------------------------------------------------+
//|  foreach pseudo-操作符                                           |
//+------------------------------------------------------------------+
#define ForEach(index, array) for (int index = 0,                     \
   max_##index=ArraySize((array));                                    \
   index<max_##index; index++)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string array[]={"12","23","34","45"};
//--- 使用ForEach绕过数组
   ForEach(i,array)
     {
      PrintFormat("%d: array[%d]=%s",i,i,array[i]);
     }
  }
//+------------------------------------------------------------------+
/* Output result
   0: array[0]=12
   1: array[1]=23
   2: array[2]=34
   3: array[3]=45
*/
```

© 2000-2025, MetaQuotes Ltd.

对于编译器来说，所有这三个#define指令行看起来像一条很长的线。上面的示例也适用于##字符，即在 #define宏中使用合并操作符将两个宏令牌合并为一个。在宏定义中令牌合并操作符既不能是第一个也不能是最后一个。

© 2000-2025, MetaQuotes Ltd.

# 常量声明 (#define, #undef)

编译器使用预处理程序指令在编译之前对源代码进行预处理。这个指令通常以#开始，因此编译器禁止在变量，函数等名称中使用符号。

每个指令都由一个单独的条目描述，并且在换行符之前都是有效的。在一个条目中不能使用多个指令。如果指令条目太大，可以使用'\'符号将其分隔成若干行。 在这种情况下，下一行被认为是指令条目的延续。

#define直接用来指定常量助记名。有两种形式：

```
#define identifier expression                    // 无参数形式
#define identifier(par1,... par8) expression     // 参数形式
```

#define 直接代替 expression 为在源文件中找出更多的*标识符*目录。*标识符*仅代表自己时可被替换。如果*标识符*是注解一部分，字符串一部分或者其他标识符一部分，则不能代替。

常量标识符与变量名同样规则。值可以是多种类型：

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

expression 由几种标记组成，例如关键字，常量，有常量无常量表达式。expression在直线末端结束，不可转到下一行。

示例：

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

## 参数形式 #define

通过参数形式，所有随后发现的标识符条目都将被考虑实际参数的表达式所取代。例如：

© 2000-2025, MetaQuotes Ltd.

```
 // a和b两个参数的示例
#define A 2+3
#define B 5-1
#define MUL(a, b) ((a)*(b))

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

使用参数表达式时一定要在圆括号中附上参数，这将有助于避免不易发现的不明显的错误。如果我们不使用括号重写代码，结果将会有所不同：

```
 // a和b两个参数的示例
#define A 2+3
#define B 5-1
#define MUL(a, b) a*b

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

使用参数形式时，最多允许8个参数。

```
 // 正确的参数形式
#define LOG(text)  Print(__FILE__,"(",__LINE__,") :",text)   // 一个参数 - 'text'

 // 不正确的参数形式
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9)   p1+p2+p3+p4 // 从p1到p9的超过8个
```

## #undef指令

定义之前，#undef指令取消宏替换声明。

示例：

```
#define MACRO

void func1()
```

© 2000-2025, MetaQuotes Ltd.

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

#undef MACRO

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

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

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

另见

标识符, 字符常量

© 2000-2025, MetaQuotes Ltd.

# 程序属性（#property）

每个mql5-程序都允许指定额外的名为#property的特殊参数，有助于客户端不用启动程序就能够进行贴切的互联网服务。首先，它包括指标外部设置。包括文件中描述的属性完全忽略。属性必须在mql5-主文件中指定。

```
#property identifier value
```

编译器在执行模块配置中编写声明值。

| 常数 | 类型 | 描述 |
| --- | --- | --- |
| icon | 字符串 | 将用作EX5程序图标的图形文件路径。路径规范规则与资源规则一样。属性必须指定在MQL5源代码的主模块。图标文件必须是 ICO 格式。 |
| link | 字符串 | 链接到公司网站 |
| copyright | 字符串 | 公司名 |
| version | 字符串 | 程序版本，最大31位字符 |
| description | 字符串 | mql5-程序的简单文本描述。现存几种*描述*，每个描述文本中的一行。包括换行所有*描述*总长度不能超过511个字符。 |
| stacksize | int | 循环调用堆栈大小 |
| library | | 程序库；指定非启动函数，带有输出修饰符的函数也可以引入到其他mql5-程序中 |
| indicator_applied_price | int | 指定默认值到“应用到”字段。如果属性没有指定，您可以指定一个 ENUM_APPLIED_PRICE值，默认值为PRICE_CLOSE |
| indicator_chart_window | | 显示图表窗口指标 |
| indicator_separate_window | | 显示独立窗口指标 |
| indicator_height | int | 修正指标子窗口的像素高度（属性 INDICATOR_HEIGHT） |
| indicator_buffers | int | 指标计算缓冲区 |
| indicator_plots | int | 指标图解系列数目 |
| indicator_minimum | 双精度 | 独立指标窗口底部比例限制 |
| indicator_maximum | 双精度 | 独立指标窗口顶部比例限制 |
| indicator_labelN | 字符串 | 为数据窗口显示的N-th 图解系列设置标签。对于需要多指标缓冲（DRAW_CANDLES, DRAW_FILLING and others）的图解系列，通过分号“;”规定标签名。 |

© 2000-2025, MetaQuotes Ltd.

| 常数 | 类型 | 描述 |
| --- | --- | --- |
| indicator_colorN | color | N行颜色，N为图解系列的数量；从1开始计数 |
| indicator_widthN | int | 图解系列粗线条，N为图解系列数量，从1开始计数 |
| indicator_styleN | int | 图解系列 的线型，由ENUM_LINE_STYLE 值指定。N为图解系列数量，从1开始计数 |
| indicator_typeN | int | 绘图种类，由ENUM_DRAW_TYPE值指定。N为图解系列数量，从1开始计数 |
| indicator_levelN | 双精度 | 独立指标窗口中N水平线 |
| indicator_levelcolor | color | 指标水平线颜色 |
| indicator_levelwidth | int | 指标粗水平线 |
| indicator_levelstyle | int | 指标水平线类型 |
| script_show_confirm | | 运行脚本前显示确认窗口 |
| script_show_inputs | | 运行脚本前显示属性窗口，禁止确认窗口 |
| tester_indicator | 字符串 | 以 *"Indicator_name.ex5"*形式命名自定义指标。如果相关参数通过常量字符串设置，自动定义调用iCustom()函数需要测试的指标。对于其他情况下（使用函数IndicatorCreate()或者使用无常量字符串的参数设置指标名）属性是必须的 |
| tester_file | 字符串 | 双引号中（作为常数字符串）扩展指示测试文件名。指定的文件传送到测试。若需要的话，测试的输入文件必须被指定。 |
| tester_library | 字符串 | 双引号中扩展程序库名。程序库有dll或者ex5扩展名。需要测试的程序库自动定义。然而，如果任何程序库用于自定义指标，属性是必须的。 |
| tester_set | 字符串 | 带有输入参数的值和步骤的设置文件的名称。在测试和优化之前，这个文件被传递到测试器。作为一个常数字符串，文件名以一个扩展名和双引号来指定。<br>如果您在设置文件的名称中将EA名称和版本号指定为"<expert_name>_<number>.set"，那么它会被自动添加到<number>版本号下的参数版本的下载菜单。例如，"MACD Sample_4.set"名称意味着这是一个版本号为4的"MACD Sample.mq5"EA设置文件。<br>若要研究这种格式，我们建议您在策略测试中手动保存测试/优化设置，然后打开以这种方式创建的设置文件。 |
| tester_no_cache | 字符串 | 当执行优化时，策略测试会将所执行传递的所有结果保存到优化缓存，在这里测试结果被保存在每组输入参数中。这允许 |

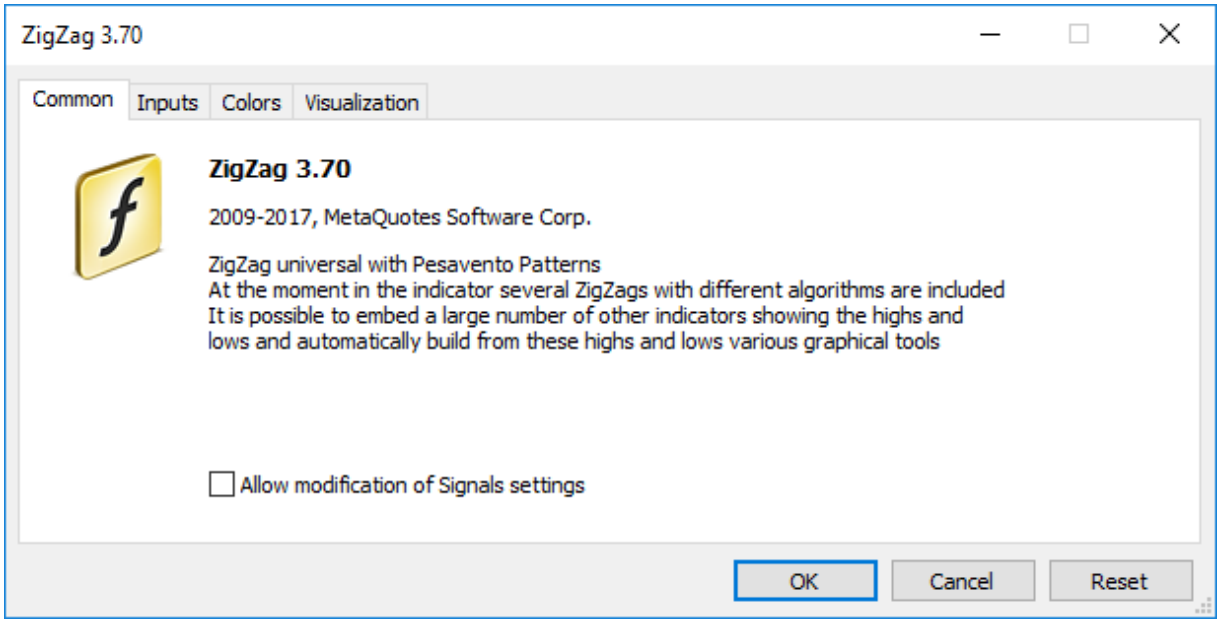
© 2000-2025, MetaQuotes Ltd.

| 常数 | 类型 | 描述 |
| --- | --- | --- |
| | | 在对相同参数进行重新优化期间使用现有的结果，而不会浪费时间重新计算。<br><br>但是在一些任务中（例如，在数学计算中），则可能必须进行计算，而无论优化缓存中是否提供现有结果。在这种情况下，该文件应该包括*tester_no_cache*属性。测试结果仍被存储在缓存中，因此您可以在策略测试中看到关于已执行传递的所有数据。 |
| tester_everytick_calculate | 字符串 | 在策略测试中，只有访问指标数据时才会计算指标，即当请求指标缓冲区的值时。如果您不需要获得每个报价的指标值，那么这可以极大地加快测试和优化的速度。<br><br>通过指定*tester_everytick_calculate*属性，您可以在每次报价时启用强制指标计算。<br><br>策略测试中的指标也可以在下列情况下在每次报价时强制计算：<br>• 当在可视模式下测试时；<br>• 如果指标具有以下任何函数：EventChartCustom、OnChartEvent、OnTimer；<br>• 如果指标是使用低于1916版本号的编译器进行创建的。<br><br>这项功能只应用在策略测试中，而程序端指标则总是在每次收到报价时进行计算。 |
| optimization_chart_mode | 字符串 | 指定将用于优化结果可视化的两个输入参数的图表类型和名称。例如，“3d、InpX、InpY”表示结果将显示在3D图表中，坐标轴基于测试的InpX和InpY参数值。因此，该属性支持将直接在程序代码中显示优化图表和图表类型的参数规格。<br>可选项：<br>• "3d, input_parameter_name1, input_parameter_name2"表示可以旋转、放大和缩小的3D可视化图表。该图表使用两个参数构建。<br>• "2d, input_parameter_name1, input_parameter_name2"表示2D网格图表，其中的每个单元格都根据结果填充特定颜色。该图表使用两个参数构建。<br>• "1d, input_parameter_name1, input_parameter_name2"表示线性图表，其中结果按照指定参数排序。每个传递都显示为一个点。该图表基于一个参数构建。<br>• "0d, input_parameter_name1, input_parameter_name2"表示按传递结果到达时间排序的常规图表。每个传递都在图表中显示为一个点。参数指示不需要，但指定参数可用于手动切换到其他图表类型。<br>或者，您只能指示图表类型，而无需指定一个或两个输入参数。在这种情况下，程序端将选择所需的参数来显示优化图表。 |

© 2000-2025, MetaQuotes Ltd.

描述和版本号的任务样本

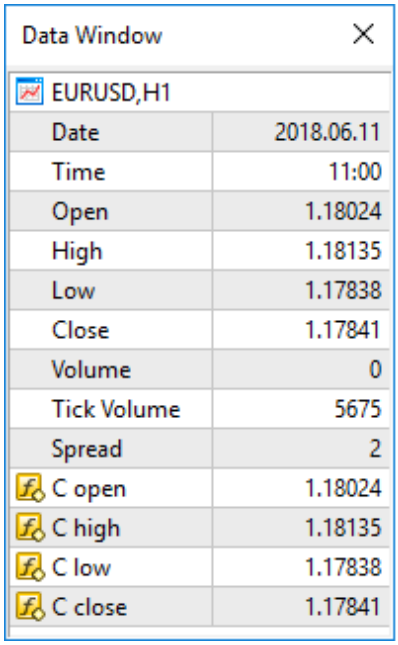
```
#property version     "3.70"      // EA交易的当前版本
#property description "ZigZag universal with Pesavento Patterns"
#property description "At the moment in the indicator several ZigZags with different a
#property description "It is possible to embed a large number of other indicators show
#property description "lows and automatically build from these highs and lows various
```




指定指标缓存独立标签示例（"C开盘价;C最高价;C最低价;C收盘价"）

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

© 2000-2025, MetaQuotes Ltd.



© 2000-2025, MetaQuotes Ltd.

# 包括文件 (#include)

#include 命令可以放置到程序的任意部分，但是通常所有文件的源代码被统一放置。调用格式：

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

示例：

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

预处理程序用 WinUser32.mqh.文件内容代替线 *#include <file_name>*。三角括号表示 WinUser32.mqh文件将会从标准目录调用(通常目录为 *terminal_installation_directory\MQL5\Include*)。当前目录不可见。

如果文件名锁在引号中，搜索将在当前目录中执行(加载的源代码主文件)。标准目录不可见。

另见

标准数据库，输入函数

© 2000-2025, MetaQuotes Ltd.

# 输入函数 (#import)

函数从MQL5编译模板(*.ex5 文件)和执行系统文件模板(文件 *.dll)通过。模板名称被指定在*#import*指令中。能够正确形成输入函数的编译器调用和组织适当的参数传送，需要带有完整的函数描述部分。函数描述会立即按照*#import* "模板名称"执行。新的*#import*命令完成引入输入函数描述部分。

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

输入函数可以有几个名称。相同名称不同的模块的函数可以同时输入。输入函数名与嵌入函数名一致。范围解析操作决定需要调用哪个函数。

搜索 #import 关键字后指定文件的命令描述在调用导入函数。

因为引入函数是在模块外面被编写，编译器无法检查通过参量的正确性。因此，为避免运行错误， 它必须精确地描述传送到输入函数的参数的组成和命令。传到输入函数的参数 (从EX5，和从DLL-模块)可以有默认值。

以下在输入函数中不能用作参数：

- 指针 (*);
- 连接动态数组或者指针的对象。

类，字符串数组或者包括字符串或者动态数组的复合对象不能作为参数传送到DLL输入函数。

示例：

```
#import "stdlib.ex5"
string ErrorDescription(int error_code);
int    RGB(int red_value,int green_value,int blue_value);
bool   CompareDoubles(double number1,double number2);
string DoubleToStrMorePrecision(double number,int precision);
string IntegerToHexString(int integer_number);
#import "ExpertSample.dll"
int    GetIntValue(int);
double GetDoubleValue(double);
string GetStringValue(string);
double GetArrayItemValue(double &arr[],int,int);
bool   SetArrayItemValue(double &arr[],int,int,double);
double GetRatesItemValue(double &rates[][6],int,int,int);
#import
```

在mql5程序执行期间输入函数，需要使用前期绑定。这就意味着ex5程序加载时启动资料库。

不建议使用*Drive:\Directory\FileName.Ext*加载模块的权限定名。MQL5资料库会从*terminal_dir\MQL5\Libraries*文件夹中加载。

© 2000-2025, MetaQuotes Ltd.

如果导入函数的调用版本不同于32位和64位Windows版本，那么这两种版本都应被导入，并且正确的函数版本应使用_IsX64变量显式调用。

例如：

```
#import "user32.dll"
//--- 对32位系统
int    MessageBoxW(uint hWnd,string lpText,string lpCaption,uint uType);
//--- 对64位系统
int    MessageBoxW(ulong hWnd,string lpText,string lpCaption,uint uType);
#import
//+------------------------------------------------------------------+
//|  MessageBox_32_64_bit 使用适当的MessageBoxW()版本                |
//+------------------------------------------------------------------+
int MessageBox_32_64_bit()
  {
   int res=-1;
   //--- 如果我们使用64位Windows
   if(_IsX64)
     {
      ulong hwnd=0;
      res=MessageBoxW(hwnd,"64-bit MessageBoxW call example","MessageBoxW 64 bit",MB_C
     }
   else  // 我们使用32位Windows
     {
      uint hwnd=0;
      res=MessageBoxW(hwnd,"32-bit MessageBoxW call example","MessageBoxW 32 bit",MB_C
     }
   return (res);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int ans=MessageBox_32_64_bit();
   PrintFormat("MessageBox_32_64_bit returned %d",ans);
  }
```

## 从.NET库导入函数

若要使用.NET程序库函数，只需导入DLL本身，而不必定义特定的函数。MetaEditor自动导入所有可以使用的函数：

- 简单结构（POD，普通旧数据）—仅包含简单数据类型的结构。
- 有参数的公共静态函数，在这里只有简单类型和POD结构或其数组可被使用。

若要从程序库调用函数，简单导入即可：

© 2000-2025, MetaQuotes Ltd.

```
#import "TestLib.dll"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int x=41;
   TestClass::Inc(x);
   Print(x);
  }
```

TestClass的Inc函数C#代码如下：

```
public class TestClass
  {
   public static void Inc(ref int x)
     {
      x++;
     }
  }
```

作为执行结果，脚本返回值为42。

另见

包括文件

© 2000-2025, MetaQuotes Ltd.

# 条件编译 (#ifdef, #ifndef, #else, #endif)

编译器使用预处理程序指令在编译之前对源代码进行预处理。这个指令通常以#开始，因此编译器禁止在变量，函数等名称中使用符号。

每个指令都由一个单独的条目描述，并且在换行符之前都是有效的。在一个条目中不能使用多个指令。如果指令条目太大，可以使用'\'符号将其分隔成若干行。 在这种情况下，下一行被认为是指令条目的延续。

预处理条件编译指令允许根据某个条件的实现编译或跳过程序的一部分。

条件可以采取下列形式之一。

```
#ifdef identifier
   // 如果#define指令已经为预处理器定义标识符那么就会编译位于这里的代码。
#endif
```

```
#ifndef identifier
   // 如果#define预处理器指令当前没有定义标识符则编译位于这里的代码。
#endif
```

任何条件编译指令可以跟在任何行数之后可能包含#else指令并以#endif结束。如果验证条件是真的，#else和#endif之间的行即被忽略。如果验证条件不满足，所有检查和#else之间的指令(或#endif指令如果前者不在)即被忽略。

例如：

```
#ifndef TestMode
   #define TestMode
#endif
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   #ifdef TestMode
      Print("Test mode");
   #else
      Print("Normal mode");
   #endif
  }
```

根据程序类型和编译模式，按以下方式定义标准宏：

__MQL5__ macro在编译*.mq5文件时定义，__MQL4__ macro在编译*.mq4文件时定义。
_DEBUG macro 在debug模式编译时定义。
_RELEASE macro 在release模式编译时定义。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
   #ifdef __MQL5__
      #ifdef _DEBUG
         Print("Hello from MQL5 compiler [DEBUG]");
      #else
        #ifdef _RELEASE
           Print("Hello from MQL5 compiler [RELEASE]");
        #endif
     #endif
   #else
      #ifdef __MQL4__
         #ifdef _DEBUG
           Print("Hello from MQL4 compiler [DEBUG]");
        #else
           #ifdef _RELEASE
              Print("Hello from MQL4 compiler [RELEASE]");
           #endif
        #endif
     #endif
   #endif
  }
```

© 2000-2025, MetaQuotes Ltd.

# 面向对象的程序设计

面向对象的程序设计(OOP)主要是针对数据或者与数据密不可分行文的程序设计。数据和行为合起来称为类，对象就是类的实例。

面向对象处理组件：

- 类型封装和扩展性
- 继承机制
- 多态性
- 重载
- 虚拟函数

OOP把计算当成行为建模。模型项是抽象计算代表的对象。假设我们想编写一个著名的游戏"Tetris"。我们必须学习如何用四个正方形随机组成的图形来建模。我们还需要调节图形下降的速度，定义图形旋转变换的操作。屏幕上图形移动受文档对象模型边框限制，这也需要建模。此外，消除填满的行，然后获得相应的得分。

因此，这个简单易懂的游戏需要创建几个模型-图形模型，文档对象模型，移动模型等等。所有这些模型都是抽象的，通过电脑运算表示。描述这些模型，抽象数据类型ADT（或者复杂数据类型）概念被使用。严格说来，DOM中"图形"运动模型并不是数据类型，但是它是使用"DOM"数据类型限制条件，"图形"数据类型全部操作。

对象是类变量。面向对象的程序设计允许您轻松创建使用ADT。面向对象程序设计使用继承机制。其优势是允许从用户已经定义的数据类型中获得衍生类型。

例如，创建俄罗斯方块图形，首先创建一个基本类图形很方便；代表所有可能图形的其他七个类可以从基本图形衍生出来。图形行为在基本类中就确定了，而执行每个独立图形的行为在衍生类中定义。

在OOP中对象对其行为负责。ADT开发人员包括描述相关对象期望的行为代码。实际上，对象对其行为负责，显著简化了对象用户程序设计的任务。

如果我们想要在屏幕上画一个图，我们需要知道中心点在哪里，以及如何画它们。如果独立的图形知道如何画自己，当使用这个图形时程序员可以发送"draw"消息。

MQL5语言类似于C++，也有ADT的封装机制。一方面，封装与内部特殊类型结合，另一方面，也与影响这类型对象的可接入外部函数连结。执行细节对于使用这个类型的程序很难达到。

OOP概念有一系列相关概念，包括以下内容：

- 模拟真实世界操作
- 用户定义数据类型的有效性
- 隐藏执行细节
- 通过继承机制重新使用代码的可能性
- 执行期解释调用函数

一些概念非常不明确，有一些很抽象，另一些又很大众化。

© 2000-2025, MetaQuotes Ltd.

# 类型的密封和扩展

OOP是编写软件的均衡办法。数据和行为放在一起。密封创建了定义用户的数据类型，扩展语言数据类型并相互作用。类型扩展是一个向语言添加用户定义数据类型的机会，该类型像基本类型一样，很容易使用。

抽象数据类型，例如，字符串，是著名行为类型理想的描述。

字符串用户知道字符串操作，例如连结或者打印，都有一定的行文。连结和打印操作称为类函数。

某种ADT执行有一定的限制，例如，字符串长度有限制。这些限制影响了行为全部开放。同时，内部或者私人执行细节不直接影响用户看对象的方法。例如，字符串尝以数组来执行，而这个数组的内部基本地址及其名称对于用户则不是必须的。

密封就是当向用户定义类型提供端口时，隐藏执行细节的能力。在MQL5中，及在C++中，类和结构定义（类和结构）用于接入关键字 private, protected和public连结的封存规定中。

关键字public 接入其后面的成员，无任何限制就可以打开。无关键字，类成员默认封锁。个人成员仅可以接入这个类里成员函数。

Protected类函数不仅可以在其类中得到，也可以在其继承类中得到。Public类函数可以在类声明范围任一函数中得到。保护可以隐藏类的部分执行，因此阻止了数据结构中意外改变。接入限制和数据隐藏都是面向对象程序设计的一个特点。

通常情况下，类函数由protected修饰符保护和声明，读取和编写值通过使用publics接入修饰符定义的所谓的特别的set-and get-类函数来执行。

示例：

```
class CPerson
  {
protected:
   string            m_name;                       // 名称
public:
   void              SetName(string n){m_name=n;}// 设置 名称
   string            GetName(){return (m_name);} // 返回 名称
  };
```

这个办法提供了几种优势。首先，通过函数名我们知道它是做什么的-设置或者获得类成员值。其次，也许在将来我们将需要改变CPerson类中或者任一衍生类中的m_name CPerson变量类型。

在这种情况下，我们只需要改变执行函数SetName()和GetName()，而CPerson类的对象可用于使用无代码改变的程序因为用户甚至不知道m_name数据类型已经改变。

示例：

```
struct Name
  {
   string            first_name;                   // 名称
   string            last_name;                    // 上一个 名称
```

© 2000-2025, MetaQuotes Ltd.

```
  };

class CPerson
  {
protected:
   Name              m_name;                       // 名称
public:
   void              SetName(string n);
   string            GetName(){return(m_name.first_name+" "+m_name.last_name);}
private:
   string            GetFirstName(string full_name);
   string            GetLastName(string full_name);
  };

void CPerson::SetName(string n)
  {
   m_name.first_name=GetFirstName(n);
   m_name.last_name=GetLastName(n);
  }

string CPerson::GetFirstName(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>0) StringSetCharacter(full_name,pos,0);
   return(full_name);
  }

string CPerson::GetLastName(string full_name)
  {
   string ret_string;
   int pos=StringFind(full_name," ");
   if(pos>0) ret_string=StringSubstr(full_name,pos+1);
   else      ret_string=full_name;
   return(ret_string);
  }
```

另见

数据类型

© 2000-2025, MetaQuotes Ltd.

# 继承算法

OOP属性特点是通过继承算法鼓励代码重复使用。新类从现存的，基本类中生成。衍生类使用基本类成员，但是也做以更改和补充。

许多类型是现存类型的变异。为每个类开发新代码非常乏味。此外，新代码意味着新的错误。衍生类继承了基本类的描述，因此不必重复开发和重复测试代码。继承关系是层级体系。

层级是一个允许复制所有多样性和复杂性元素的类函数。它介绍了对象等级。例如，元素周期表有气体。它们对所有的周期元素控制固有的属性。

惰性气体是下一重要子集的 构成。层级就是惰性气体，例如氩是气体，是系统的一部分。这样的层级可以很轻松的解释惰性气体。我们知道其原子包括质子和电子，对于所有其他元素也是如此。

我们知道如其他气体一样它们在常温也是气体状态。我们知道惰性气体子集中，无气体能与其他化学元素发生化学反应，它是所有惰性气体的特性。

考虑到几何图形的继承例子。描述各种简单图形 (圆形，三角形，矩形，正方形等等)，最好的方法就是创建基本类 (ADT)，它是所有衍生类的先祖。

让我们创建一个基本类 CShape，它包括描述图形最常用的构件。这些构件描述任何图形所特有的属性-图像类型和定位点主坐标。

示例：

```
//--- 基本类的形状
class CShape{}
  {
protected:
   int       m_type;                    // 形状类型
   int       m_xpos;                    // 基本点的X - 坐标
   int       m_ypos;                    // 基本点的Y - 坐标
public:
             CShape(){m_type=0; m_xpos=0; m_ypos=0;} // constructor
   void      SetXPos(int x){m_xpos=x;} // 设置 X
   void      SetYPos(int y){m_ypos=y;} // 设置 Y
  };
```

下一步，创建基本类衍生出的新类，这里我们可以添加说明类的必要的字段。对于圆形添加包括半径值构件是必须的。正方形以边值为特点。因此，由继承基本类CShape而衍生的类如下声明：

```
//--- 派生类 圆形
class CCircle : public CShape         // 冒号后定义基本类
  {                                   // 从继承算法开始
private:
   int             m_radius;          // 圆弧半径

public:
                   CCircle(){m_type=1;}// 构造函数, 类型 1
  };
```

© 2000-2025, MetaQuotes Ltd.

正方形类声明类似：

```
//--- 派生类 方形
class CSquare : public CShape         // 冒号后定义基本类
  {                                     // 从继承算法开始
private:
   int            m_square_side;       // 方形的边

public:
                  CSquare(){m_type=2;} // 构造函数，类型2
  };
```

注意对象创建时首先调用基本类构造函数，然后调用衍生类的构造函数。当对象毁坏时首先调用衍生类的析构函数，然后调用基本类析构函数。

因此，通过声明基本类中常用构件，我们可以在衍生类中添加额外构件，指定特殊类。继承算法允许创建多次重复使用的强大的代码函数库。

从现存类创建衍生类的句法如下：

```
class class_name :
          (public | protected | private) opt  base_class_name
  {
    class members declaration
  };
```

衍生类一方面就是其构件的可见性（公开），继承人（继承）。关键字public，protected 和private用于指定范围，该范围中基本类构件也对衍生类有效。衍生类表头中冒号后的Public关键字表明基本类CShape的protected和public构件应该继承为衍生类CCircle的protected和public构件。

基本类的private类构件对衍生类无效。public继承也意味着衍生类（CCircle and CSquare）就是CShapes。也就是，正方形(CSquare)是一个图形(CShape)，但是图形却不一定就是正方形。

衍生类是基本类的变体，它继承了基本类的protected 和 public构件。基本类的构造函数和析构函数不能继承。除了基本类的构件，新构件也会添加进衍生类中。

不同于基本类，衍生类包括执行构件函数。与重载无共同点，同名函数的意思会因签名不同而不同。

在protected继承中，基本类的public和protected构件成为衍生类的protected构件。在private 继承中，基本类的public 和protected构件成为衍生类的private构件。

在protected 和 private 继承中，“衍生类对象就是基本类对象”的关系不是真的。protected 和 private 继承类型也很少见，每一个都需要小心使用。

应该了解继承类型(public, protected或private)不会影响**从衍生类访问继承层次结构的基类成员**的方法。任何继承类型中，只有public 和 protected访问说明符声明的基类成员可用衍生类。让我们考虑一下下面的示例：

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"
//+------------------------------------------------------------------+
//| 一些访问类型的示例类                                             |
//+------------------------------------------------------------------+
class CBaseClass
  {
private:              //--- 从衍生类 private成员不可用
   int                m_member;
protected:            //--- 从基类及其衍生类 protected方法不可用
   int                Member(){return(m_member);}
public:               //--- 类构造函数可用于类的所有成员
                      CBaseClass(){m_member=5;return;};
private:              //--- 将值分配给m_member的private方法
   void               Member(int value) { m_member=value;};

  };
//+------------------------------------------------------------------+
//| 有错误的衍生类                                                   |
//+------------------------------------------------------------------+
class CDerived: public CBaseClass // 由于默认的原因，public继承规范可以忽略。
  {
public:
   void Func() // 在衍生类中，定义调用到基类成员的函数。
     {
      //--- 试图修改基类的private成员
      m_member=0;          // 错误，基类private成员不可用
      Member(0);           // 错误，基类的private方法不可用在衍生类
      //--- 阅读基类的成员
      Print(m_member);     // 错误，基类的private成员不可用
      Print(Member());     // 没有错误，protected方法可从基类及其衍生类使用
     }
  };
```

在上面的示例中，CBaseClass只有一个public方法——构造函数。当创建类对象时构造函数会自动调用。因此，private成员m_member和protected方法Member()不能从外部调用。但是如果是public继承类型，基类的Member()方法将可以从衍生类使用。

如果是protected继承类型的情况下，所有public和protected访问权限的基类成员都会成为protected。这就意味着如果public基类的数据成员和方法从外部访问，那么protected继承类型的情况下，他们只能从衍生类及其衍生品的类中使用。

```
//+------------------------------------------------------------------+
//| 一些访问类型的示例类                                             |
//+------------------------------------------------------------------+
class CBaseMathClass
  {
private:              //--- private成员不可从衍生类使用
   double             m_Pi;
public:               //--- 获取和设置m_Pi值
   void               SetPI(double v){m_Pi=v;return;};
```

© 2000-2025, MetaQuotes Ltd.

```
   double            GetPI(){return m_Pi;};
public:              // 类构造函数可用于所有成员
                     CBaseMathClass() {SetPI(3.14);  PrintFormat("%s",__FUNCTION__);};
  };
//+------------------------------------------------------------------+
//| 一个衍生类 在此m_Pi不能修改                                      |
//+------------------------------------------------------------------+
class CProtectedChildClass: protected CBaseMathClass // Protected继承类型
  {
private:
   double            m_radius;
public:              //--- 衍生类中的Public方法
   void              SetRadius(double r){m_radius=r; return;};
   double            GetCircleLength(){return GetPI()*m_radius;};
  };
//+------------------------------------------------------------------+
//| 脚本启动函数                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建衍生类时，基类的构造函数将自动调用
   CProtectedChildClass pt;
//--- 指定半径
   pt.SetRadius(10);
   PrintFormat("Length=%G",pt.GetCircleLength());
//--- 如果评论下面的字符串，我们在编译阶段将得到一个错误，因为SetPi()现在是protected的类型
// pt.SetPI(3);

//--- 现在声明基类变量，尝试将Pi常量设置等于10
   CBaseMathClass bc;
   bc.SetPI(10);
//--- 下面是结果
   PrintFormat("bc.GetPI()=%G",bc.GetPI());
  }
```

实例表明，基类CBaseMathClass的方法SetPI()和GetPi()是开放状态并可以从程序的任何地方进行调用。但与此同时，对于从其衍生的CProtectedChildClass来说，这些方法只能从CProtectedChildClass类或其衍生类的方法进行调用。

如果是private继承类型，所有public和protected访问权限的基类成员都会成为private，并且在进一步继承中无法进行调用。

MQL5没有多继承算法。

另见

架构和类

© 2000-2025, MetaQuotes Ltd.

# 多态性

多态性为不同分类对象提供了一个机会，通过继承，当调用相同函数元素时以不同的方式回应，通过帮助创造普遍原理表述行为不仅是基础类别，更是继承类别。

继续研发基础分类CShape，定义成员函数GetArea()，目的在于计算阴影区域，在所有继承分类里，通过从基础分类继承下来，我们重新定义这种函数以计算特殊阴影面积的规则相同

在一个区域内（CSquare分类），通过面积计算一圈（CCircle分类），区域通过半径等表示。创建数组存储CShape类型交易对象，基础类别中的量个交易品种和继承的交易分类都能存储，然后就可以调用数组中每个元素的任意相同函数。

示例：

```
//--- 基本类
class CShape
  {
protected:
   int            m_type;                 // 形状类型
   int            m_xpos;                 // 基本点的X - 坐标
   int            m_ypos;                 // 基本点的Y - 坐标
public:
   void           CShape(){m_type=0;};    // 构造函数, 类型=0
   int            GetType(){return(m_type);};// 返回 图形类型
virtual
   double         GetArea(){return (0); }// 返回 图形区域
  };
```

现在，所有的派生类都有成员函数 getArea()，返回0值，该函数以每个后代变化转变为基础实现。

```
//--- 派生类 圆形
class CCircle : public CShape              // 冒号后定义基本类
  {                                        // 从继承算法开始
private:
   int            m_radius;               // 圆弧半径

public:
   void           CCircle(){m_type=1;};   // 构造函数, 类型=1
   void           SetRadius(double r){m_radius=r;};
   virtual double GetArea(){return (3.14*m_radius*m_radius);}// 圆形 区域
  };
```

对于Square分类来说，申报是相同的：

```
//--- 派生类 方形
class CSquare : public CShape              // 冒号后定义基本类
  {                                        // 从继承算法开始
private:
   int             m_square_side;         // 方形的边
```

© 2000-2025, MetaQuotes Ltd.

```
public:
   void             CSquare(){m_type=2;}; // 构造函数, 类型=1
   void             SetSide(double s){m_square_side=s;};
   virtual double GetArea(){return (m_square_side*m_square_side);}// 方形区域
  };
```

为了计算圆圈面积，需要类似m_radius和m_square_side类型值，因此在类似分类中添加SetRadius和SetSide()

假设从一个基本类型CShape衍生的不同类型（CCircle和CSquare）的对象用在我们的程序。多态性允许创建基类CShape对象的数组，但是当声明该数组时，则这些对象身份不明且它们的类型也不明确。

有关每个数组元素将包括的对象类型决策将在程序执行期间直接采用。这包括动态创建合适的类的对象，因此需要使用对象指针来替代对象。

new操作符用于动态创建对象。每个这样的对象都必须独立且明确地使用delete操作符删除。所以，我们将声明CShape类型的指针数组，并为每个元素(new Class_Name)创建合适类型的对象，如下面的脚本示例所示：

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明基本类型对象指针的数组
   CShape *shapes[5];   // CShape对象指针的数组

//--- 这里以衍生对象填充数组
//--- 声明指针到CCircle类型的对象
   CCircle *circle=new CCircle();
//--- 在圆形指针设置对象属性
   circle.SetRadius(2.5);
//--- 将指针值置于shapes[0]
   shapes[0]=circle;

//--- 创建另一个CCircle对象并写下其指针在shapes[1]
   circle=new CCircle();
   shapes[1]=circle;
   circle.SetRadius(5);

//--- 这里我们故意“忘记”设置shapes[2]值
//circle=new CCircle();
//circle.SetRadius(10);
//shapes[2]=circle;

//--- 不使用的元素设置为NULL
   shapes[2]=NULL;

//--- 创建CSquare对象并写下其指针在shapes[3]
```

© 2000-2025, MetaQuotes Ltd.

```
   CSquare *square=new CSquare();
   square.SetSide(5);
   shapes[3]=square;

//--- 创建CSquare对象并写下其指针到shapes[4]
   square=new CSquare();
   square.SetSide(10);
   shapes[4]=square;

//--- 我们有一组指针，获得其大小
   int total=ArraySize(shapes);
//--- 通过数组的所有指针循环通过
   for(int i=0; i<5;i++)
     {
      //--- 如果指定指数的指针有效
      if(CheckPointer(shapes[i])!=POINTER_INVALID)
        {
         //--- 记录类型和正方形
         PrintFormat("The object of type %d has the square %G",
               shapes[i].GetType(),
               shapes[i].GetArea());
        }
      //--- 如果指针类型POINTER_INVALID
      else
        {
         //--- 错误的通知
         PrintFormat("Object shapes[%d] has not been initialized! Its pointer is %s",
                     i,EnumToString(CheckPointer(shapes[i])));
        }
     }

//--- 我们必须删除所有已创建的动态对象
   for(int i=0;i<total;i++)
     {
      //--- 我们只可以删除POINTER_DYNAMIC类型指针的对象
      if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
        {
         //--- 删除的通知
         PrintFormat("Deleting shapes[%d]",i);
         //--- 通过指针删除对象
         delete shapes[i];
        }
     }
  }
```

请注意，当使用delete操作符删除对象时，必须检查指针类型。只有POINTER_DYNAMIC 指针对象可以使用delete删除。对于其他类型的指针，将返回错误。

© 2000-2025, MetaQuotes Ltd.

但此外，通过继承和多态性重新定义函数，包括一个工具和一个分离中建立的相同函数的不同参量，这表示该分类拥有几个相同名称的函数，但是不同类型的参量，因此，通过function overload实施多态性。

另见

标准程序库

© 2000-2025, MetaQuotes Ltd.

# 重载

在一个类中定义两个或者以上同名类函数是可能的，但是会有不同数量的参量。当这个发生时，类函数就称为*重载*而这样的过程被称为*类函数重载*。

类函数重载是一种实现多态的方法。如函数重载一样类函数重载按照以下规则执行。

如果调用的函数无准确匹配，编译器就会在三个连续阶中搜索适合的函数：

1. 类函数中搜索；
2. 基本类函数中搜索，按照最近到最开始顺序。
3. 其他函数中搜索。

如果任何阶中都没有确切匹配的，但是在不同阶中找到几个适合的函数，使用最近阶找到的函数。一个阶中，只有一个适合的函数。

MQL5没有操作符重载。

另见

重载函数

© 2000-2025, MetaQuotes Ltd.

# 虚拟函数

虚拟关键字是函数说明符，会根据原理选择动态的运行时间，并在基本或派生类别中找出恰当函数。架构中不能有虚拟函数，它只能用于改变函数声明 。

虚拟函数，与一般函数一样，都需要 可执行体。当调用时，语义与其他函数一样

虚拟函数在派生类里可能会遭到拒绝，选择函数定义的种类需要虚拟函数的动态调用 (在运行时间里)，典型情况是，当基本类别包含虚拟函数时，派生类都含有函数里特有的版本。

指针指向基本类别里可以标明基本分类对象或者派生类对象，函数成员选择的调用会在运行时间内进行，会依靠类型对象执行，并不是指标的类型，如果没有派生类成员，数据分类的虚拟函数就是默认值。

无论是否用虚拟关键字描述，析构函数永远是虚拟的。

在MT5_Tetris.mq5例子中虚拟函数的使用，虚拟函数 Draw 的基本类别CTetrisShape，定义在包含文件 MT5_TetrisShape.mqh.中。

```
//+------------------------------------------------------------------+
class CTetrisShape
  {
protected:
   int               m_type;
   int               m_xpos;
   int               m_ypos;
   int               m_xsize;
   int               m_ysize;
   int               m_prev_turn;
   int               m_turn;
   int               m_right_border;
public:
   void              CTetrisShape();
   void              SetRightBorder(int border) { m_right_border=border; }
   void              SetYPos(int ypos)          { m_ypos=ypos;           }
   void              SetXPos(int xpos)          { m_xpos=xpos;           }
   int               GetYPos()                  { return(m_ypos);        }
   int               GetXPos()                  { return(m_xpos);        }
   int               GetYSize()                 { return(m_ysize);       }
   int               GetXSize()                 { return(m_xsize);       }
   int               GetType()                  { return(m_type);        }
   void              Left()                     { m_xpos-=SHAPE_SIZE;    }
   void              Right()                    { m_xpos+=SHAPE_SIZE;    }
   void              Rotate()                   { m_prev_turn=m_turn; if(++m_turn>3) r
   virtual void      Draw()                     { return;                }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

© 2000-2025, MetaQuotes Ltd.

进一步说，对于每一个分类，该函数都会依照下一代分类的特性来执行，例如，第一个模型CTetrisShape1有自己的执行函数Draw()：

```
class CTetrisShape1 : public CTetrisShape
  {
public:
   //--- 绘画形状
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      if(m_turn==0 || m_turn==2)
        {
         //--- 水平线
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
           }
        }
      else
        {
         //--- 垂直线
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos);
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+i*SHAPE_SIZE);
           }
        }
     }
  }
```

方形台由CTetrisShape6分类描述，并实现Draw()语法：

```
class CTetrisShape6 : public CTetrisShape
  {
public:
   //--- 绘画形状
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      for(i=0; i<2; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
```

© 2000-2025, MetaQuotes Ltd.

```
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
        }
      for(i=2; i<4; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+(i-2)*SHAPE_SIZE);
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+SHAPE_SIZE);
        }
     }
  };
```

根据分类，建立对象归属，在派生类里调用虚拟函数。

```
void CTetrisField::NewShape()
  {
//--- 创建7个随机形状中的一个
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- 绘画
   m_shape.Draw();
//---
  }
```

## 修饰符 'override'

'override' 修饰符表示声明的函数必须重写父类的函数类。通过这个函数类您可以避免重写错误，例如它可以让您避免意外修改函数类签名。假设，'func' 函数类定义在基本类。函数类会接受int变量为自变量：

```
class CFoo
  {
   void virtual func(int x) const { }
  };
```

下一步，函数类重写在子类：

```
class CBar : public CFoo
  {
   void func(short x) { }
  };
```

然而，自变量类型错误的从int 变为short。实际上，这不是类函数重写，而是类函数重载。根据重载函数定义算法，在某些情况下编译器可以选择在基类中定义的函数类来替代重写函数类。

© 2000-2025, MetaQuotes Ltd.

为了避免这种错误，您应该明确添加 'override' 修饰符到您想重写的函数类。

```
class CBar : public CFoo
  {
   void func(short x) override { }
  };
```

如果在重写过程中更改了函数类签名，编译器将无法再父类中找到相同签名的函数类，并将返回一个编译错误：

```
'CBar::func' 函数类是通过'override' 标识符声明但不会重写任何基本 类 的类函数
```

## 修饰符 'final'

'final' 修饰符是相反的 — 它禁止函数类在子类中重写。如果函数类的实现非常充分并得到完全完成，则以 'final' 修饰符声明这个函数类以确保其以后不被修改。

```
class CFoo
  {
   void virtual func(int x) final { }
  };

class CBar : public CFoo
  {
   void func(int) { }
  };
```

如果您像上面示例一样试图以 'final' 修饰符重写函数类，编译器将返回一个错误：

```
声明为'final'的'CFoo::func' 函数类不能通过 'CBar::func'重写
请见 'CFoo::func' 声明
```

另见

标准程序库

© 2000-2025, MetaQuotes Ltd.

# 类/结构的静态成员

## 静态成员

类成员可以使用存储类修饰符static进行声明。这些数据成员通过该类的所有实例共享并存储在一个地方。为每个类对象变量创建非静态数据成员。

无法声明类的静态成员会导致需要在程序的全局层面声明这些数据。它会破坏数据及其类之间的关系，且不符合OOP的基本范式-加入数据和在类中处理它们的方法。静态成员允许类数据不特定于具体实例，存在于类的范围。

由于静态类成员不依赖于具体实例，则对它的引用如下：

```
class_name::variable
```

这里 *class_name* 是类的名称，而 *variable* 是类成员的名称。

如您所见，如果访问类的静态成员，则会使用内容解析运算符 :: 。 当您在类方法访问静态成员，内容操作符为可选项。

类的静态成员必须以所需的值进行显式初始化。为此，它必须在全局范围声明和初始化。静态成员初始化的顺序对应其在全局范围声明的顺序。

例如，我们有一个用于解析文本的*CParser* 类，并且我们需要统计处理单词和字符的总数。我们只需将必要的类成员声明为静态并在全局层面给予初始化。然后类的所有实例都将使用常见的单词和字符计数器。

```
//+------------------------------------------------------------------+
//| 类"文本解析"                                                     |
//+------------------------------------------------------------------+
class CParser
  {
public:
   static int        s_words;
   static int        s_symbols;
   //--- 构造函数和析构函数
                     CParser(void);
                    ~CParser(void){};
  };
...
//--- 全局层面解析类静态成员的初始化
int CParser::s_words=0;
int CParser::s_symbols=0;
```

静态类成员可以通过 *const*关键字来声明。这种静态常量必须在全局层面以*const* 关键字进行初始化：

```
//+------------------------------------------------------------------+
//| 类"Stack" 存储处理数据                                           |
//+------------------------------------------------------------------+
class CStack
  {
public:
                     CStack(void);
                    ~CStack(void){};
```

© 2000-2025, MetaQuotes Ltd.

```
...
private:
   static const int  s_max_length; // 最大存储栈能力
  };

//--- 初始化CStack类的静态常量
const int CStack::s_max_length=1000;
```

## 指针 this

关键字 this 表示一个隐式声明的其本身的指针 -到类的特定实例，执行方法的快捷菜单。它只可以使用在非静态类方法。指针 this 是任何类的隐式非静态成员。

在静态函数中您只可以访问静态成员/类方法。

## 静态方法

在MQL5可以使用静态 类型成员函数。在类内部的声明，*静态* 修饰符必须在函数返回类型之前。

```
class CStack
  {
public:
   //--- 构造函数和析构函数
                     CStack(void){};
                    ~CStack(void){};
   //--- 最大堆栈能力
   static int        Capacity();
private:
   int               m_length;      // 存储栈中的元素数量
   static const int  s_max_length; // 最大存储栈能力
  };
//+------------------------------------------------------------------+
//| 返回堆栈中存储的元素的最大数量                                   |
//+------------------------------------------------------------------+
int CStack::Capacity(void)
  {
   return(s_max_length);
  }
//--- 初始化CStack 类的静态常量
const int CStack::s_max_length=1000;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明 CStack 类型变量
   CStack stack;
//--- 调用对象的静态方法
   Print("CStack.s_max_length=",stack.Capacity());
//--- 它也可以按以下方式调用，因为方法是静态的，无需对象的存在
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("CStack.s_max_length=",CStack::Capacity());
  }
```

const 修饰符方法被称为常量并且不能修改类的隐式成员。声明类的常量函数和常量参数被称为*const-correctness*控制。通过这个控制，可以保证，编译器将确保对象值的一致性并且如果有什么问题，在编译过程中将会返回一个错误。

参数列表进入类声明以后才会放置const 修饰符。类以外的定义也应该包括 *const* 修饰符：

```
//+------------------------------------------------------------------+
//|  "矩形"类                                                        |
//+------------------------------------------------------------------+
class CRectangle
  {
private:
   double            m_width;       // 宽度
   double            m_height;      // 高度
public:
   //--- 构造函数和析构函数
                     CRectangle(void):m_width(0),m_height(0){};
                     CRectangle(const double w,const double h):m_width(w),m_height(h)
                    ~CRectangle(void){};
   //--- 计算区域
   double            Square(void) const;
   static double     Square(const double w,const double h);// { return(w*h); }
  };
//+------------------------------------------------------------------+
//| 返回 "矩形"对象区域                                              |
//+------------------------------------------------------------------+
double CRectangle::Square(void) const
  {
   return(Square(m_width,m_height));
  }
//+------------------------------------------------------------------+
//| 返回两变量的产品                                                 |
//+------------------------------------------------------------------+
static double CRectangle::Square(const double w,const double h)
  {
   return(w*h);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建等于5和6的矩形
   CRectangle rect(5,6);
//--- 用常量方法找出矩形区域
   PrintFormat("rect.Square()=%.2f",rect.Square());
//--- 通过类CRectangle的静态方法找出产品数量
   PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5));
  }
```

通过常量控制新增的参数实际就是在这种情况下，编译器生成一个特殊的优化，例如。 在只读存储器放置常量对象。

静态函数不能使用 const 修饰符定义，因为当调用该函数时，该修饰符能够确保实例成员的恒性。但是，如上所述，静态函数不能访问非静态类成员。

© 2000-2025, MetaQuotes Ltd.

另见

静态变量, 变量, 参考。修饰符 & 和关键字 this

© 2000-2025, MetaQuotes Ltd.

# 函数模板

重载函数通常被用于执行各种数据类型的类似的操作。ArraySize() 是MQL5中这种函数的一个简单例子。它返回任何类型的数组大小。实际上，是由应用程序开发人员重载这个系统函数并且隐藏重载的整个实施：

```
int  ArraySize(
   void&  array[]      // 检查数组
   );
```

它意味着MQL5语言编译器每次调用这个函数都要插入必要的实施。例如，那就是整型数组如何完成的方法：

```
int  ArraySize(
   int&  array[]      // int 类型元素数组
   );
```

ArraySize() 函数可以按照以下方式展示 MqlRates 类型数组，以便以历史数据格式处理报价：

```
int  ArraySize(
   MqlRates&  array[] // 填充MqlRates 类型值的数组
   );
```

因此，使用相同函数处理不同类型非常地方便。然而，应该先执行所有初步的工作 – 应该为正确处理的所有数据类型重载必要的函数。

但，这里有更方便的解决方案。如果，每个数据类型都应执行类似的操作，那么就可以使用函数模板。在这种情况下，程序员只需要编写一个函数模板的描述。当以这种方式描述模板时，我们应该只指定一个正式参数来替代函数处理的定义的数据类型。编译器将会自动生成多种函数，以便适当处理基于用于调用函数的参数类型的每种类型。

函数模板定义以template 关键字开始，紧随其后的是尖括号内的正式参数列表。每个正式参数之前都是typename 关键字。正式的参数类型分为内置的或用户定义的类型。它们被用于：

- 指定函数参数类型，
- 指定函数返回值类型，
- 声明函数描述中的变量

模板参数的数量不能超过8个。模板定义中的每个正式参数都应该至少在函数参数列表中出现一次。每个正式参数的名称都应该是独一无二的。

下面是一个函数模板的示例，搜索任何数值类型 (整数和实数)的数组中的最高值：

© 2000-2025, MetaQuotes Ltd.

```
template<typename T>
T ArrayMax(T &arr[])
  {
   uint size=ArraySize(arr);
   if(size==0) return(0);

   T max=arr[0];
   for(uint n=1;n<size;n++)
      if(max<arr[n]) max=arr[n];
//---
   return(max);
  }
```

该模板描述了在接收到的值中找到最高值，并返回其结果的函数。牢记MQL5内置的ArrayMaximum()函数仅返回可用来找到值本身的最高值指数。例如：

```
//--- 创建数组
   double array[];
   int size=50;
   ArrayResize(array,size);
//---  填充随机值
   for(int i=0;i<size;i++)
     {
      array[i]=MathRand();
     }

//--- 找出数组最高值的持仓
   int max_position=ArrayMaximum(array);
//--- 现在，接收数组最高值本身
   double max=array[max_position];
//--- 找出发现的值
   Print("Max value = ",max);
```

因此，我们执行两步法来接收数组最高值。使用ArrayMax()函数模板，我们可以通过发送相应类型的数组获得必要类型的结果。它意味着替换两条最近的行

```
//--- 找到数组中最高值的持仓
   int max_position=ArrayMaximum(array);
//--- 现在，接收数组最高值本身
   double max=array[max_position];
```

我们现在只能使用，即时返回与发送数组相同类型的结果的模板：

```
//--- 找出最高值
   double max=ArrayMax(array);
```

在这种情况下，通过ArrayMax()函数返回的结果类型将自动匹配数组的类型。

使用typename关键字接收参数类型为一行以便创建处理各种数据类型的通用方法。我们特别举例返回数据类型的函数为一行：

© 2000-2025, MetaQuotes Ltd.

```
#include <Trade\Trade.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CTrade trade;
   double d_value=M_PI;
   int i_value=INT_MAX;
   Print("d_value: type=",GetTypeName(d_value), ",   value=", d_value);
   Print("i_value: type=",GetTypeName(i_value), ",   value=", i_value);
   Print("trade: type=",GetTypeName(trade));
//---
  }
//+------------------------------------------------------------------+
//| 类型返回为一行                                                   |
//+------------------------------------------------------------------+
template<typename T>
string GetTypeName(const T &t)
  {
//--- 返回类型为一行
   return(typename(T));
//---
  }
```

函数模板也被用于类的方法，例如：

```
class CFile
  {
   ...
public:
   ...
   template<typename T>
   uint WriteStruct(T &data);
  };

template<typename T>
uint CFile::WriteStruct(T &data)
  {
   ...
   return(FileWriteStruct(m_handle,data));
  }
```

函数模板不能使用 export, virtual 和 #import 关键字进行声明。

## 模板函数重载

重载模板函数有时可能是必需的。例如，我们有一个使用类型转换在第一个参数编写第二参数值的模板函数。MQL5不允许转换string类型到bool。我们可以亲自做 -创建重载模板函数。例如：

```
//+------------------------------------------------------------------+
//| 模板函数                                                         |
//+------------------------------------------------------------------+
template<typename T1,typename T2>
string Assign(T1 &var1,T2 var2)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   var1=(T1)var2;
   return(__FUNCSIG__);
  }
//+------------------------------------------------------------------+
//| bool+string 特殊重载                                             |
//+------------------------------------------------------------------+
string Assign(bool &var1,string var2)
  {
   var1=(StringCompare(var2,"true",false) || StringToInteger(var2)!=0);
   return(__FUNCSIG__);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int i;
   bool b;
   Print(Assign(i,"test"));
   Print(Assign(b,"test"));
  }
```

作为代码执行结果，我们可以看到Assign()模板函数已经用于int+string组，而第二次调用时重载版本已经用于bool+string组。

```
string Assign<int,string>(int&,string)
string Assign(bool&,string)
```

另见

重载

© 2000-2025, MetaQuotes Ltd.

# 模板优势

函数模板用于当您需要在不同数据类型上执行类似操作的时候，例如，寻找数组中的最大元素。使用模板的主要优势在于您无需为每个类型编制独立的重载代码。而取代声明每个类型的多个重载

```
double ArrayMax(double array[])
  {
   ...
  }
int ArrayMax(int array[])
  {
   ...
  }
uint ArrayMax(uint array[])
  {
   ...
  }
long ArrayMax(long array[])
  {
   ...
  }
datetime ArrayMax(datetime array[])
  {
   ...
  }
```

我们只需要编写一个模板函数

```
template<typename T>
T ArrayMax(T array[])
  {
   if(ArraySize()==0)
      return(0);
   uint max_index=ArrayMaximum(array);
   return(array[max_index]);
  }
```

将其用于您的编码中：

```
double high[];
datetime time[];
....
double max_high=ArrayMax(high);
datetime lasttime=ArrayMax(time);
```

这里，指定已用数据类型的形式参数*T*被替换为编译时实际的使用类型，例如编译器为每个类型自动生成一个独立的函数 - double, datetime等等。MQL5还可以使您使用该方法的全部优势开发类模板。

© 2000-2025, MetaQuotes Ltd.

## 类模板

类模板使用尖括号<>之前的template关键字进行声明，列举包含typename关键字的形式参数列表。该条目会通知编译器在执行类时它处理的是包含定义真实变量类型的*T*形式参数的普通类。例如，创建一个矢量类来存储*T*类型元素数组：

```
#define TOSTR(x) #x+" "   // 显示对象名称的宏
//+------------------------------------------------------------------+
//| 存储T- 类型元素的矢量类                                          |
//+------------------------------------------------------------------+
template <typename T>
class TArray
  {
protected:
   T                 m_array[];
public:
   //--- 构造函数默认创建一个10元素数组
   void TArray(void){ArrayResize(m_array,10);}
   //--- 创建一个指定数组大小矢量类的构造函数
   void TArray(int size){ArrayResize(m_array,size);}
   //--- 返回存储在TArray类型对象中的数据类型和数据量
   string Type(void){return(typename(m_array[0])+":"+(string)ArraySize(m_array));};
  };
```

接下来，在程序中使用不同的方法创建3个*TArray*对象来处理多种类型

```
void OnStart()
  {
   TArray<double> double_array;    // 矢量默认大小10
   TArray<int> int_array(15);      // 矢量大小15
   TArray<string> *string_array;   // TArray<string>矢量指针
//--- 创建动态对象
   string_array=new TArray<string>(20);
//--- 在日志中显示对象名称，数据类型和矢量大小
   PrintFormat("%s (%s)",TOSTR(double_array),double_array.Type());
   PrintFormat("%s (%s)",TOSTR(int_array),int_array.Type());
   PrintFormat("%s (%s)",TOSTR(string_array),string_array.Type());
//--- 完成程序之前删除动态对象
   delete(string_array);
  }
```

脚本执行结果：

```
  double_array  (double:10)
  int_array  (int:15)
  string_array  (string:20)
```

现在，我们有不同数据类型的3种矢量：double，int和string。

类模板非常适合开发容器 -专为封装任何类型其他对象而设计的对象。容器对象是已经包含特定类型对象的集合。通常，处理存储数据会立刻内置于容器。

© 2000-2025, MetaQuotes Ltd.

例如，您可以创建不允许访问数组外元素的类模板，因此也就是避免了"超范围"的严重错误。

```
//+------------------------------------------------------------------+
//| 自由访问数组元素的类                                             |
//+------------------------------------------------------------------+
template<typename T>
class TSafeArray
  {
protected:
   T                 m_array[];
public:
   //--- 默认构造函数
   void              TSafeArray(void){}
   //--- 创建指定大小数组的构造函数
   void              TSafeArray(int size){ArrayResize(m_array,size);}
   //--- 数组大小
   int               Size(void){return(ArraySize(m_array));}
   //--- 改变数组大小
   int               Resize(int size,int reserve){return(ArrayResize(m_array,size,rese
   //--- 释放数组
   void              Erase(void){ZeroMemory(m_array);}
   //--- 通过索引访问数组元素的操作符
   T                 operator[](int index);
   //--- 从数组一次性接收全部元素的赋值操作符
   void              operator=(const T  &array[]); // T 类型数组
  };
//+------------------------------------------------------------------+
//| 通过索引接收元素                                                 |
//+------------------------------------------------------------------+
template<typename T>
T TSafeArray::operator[](int index)
  {
   static T invalid_value;
//---
   int max=ArraySize(m_array)-1;
   if(index<0 || index>=ArraySize(m_array))
     {
      PrintFormat("%s index %d is not in range (0-%d)!",__FUNCTION__,index,max);
      return(invalid_value);
     }
//---
   return(m_array[index]);
  }
//+------------------------------------------------------------------+
//| 分配数组                                                         |
//+------------------------------------------------------------------+
template<typename T>
void TSafeArray::operator=(const T  &array[])
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   int size=ArraySize(array);
   ArrayResize(m_array,size);
//--- T 类型应该支持复制操作符
   for(int i=0;i<size;i++)
      m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int copied,size=15;
   MqlRates rates[];
//--- 复制报价数组
   if((copied=CopyRates(_Symbol,_Period,0,size,rates))!=size)
     {
      PrintFormat("CopyRates(%s,%s,0,%d) returned %d error code",
      _Symbol,EnumToString(_Period),size,GetLastError());
      return;
     }
//--- 创建容器并插入MqlRates值数组
   TSafeArray<MqlRates> safe_rates;
   safe_rates=rates;
   //--- 数组内索引
   int index=3;
   PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
   //--- 数组外索引
   index=size;
   PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
  }
```

请注意类声明外描述类函数时也应该使用模板声明：

```
template<typename T>
T TSafeArray::operator[](int index)
  {
   ...
  }
template<typename T>
void TSafeArray::operator=(const T  &array[])
  {
   ...
  }
```

类和函数模板允许您定义多个逗号分隔的形式参数，例如，存储"key -value"组的地图集合：

```
template<typename Key, template Value>
class TMap
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   ...
}
```

另见

函数模板，重载

© 2000-2025, MetaQuotes Ltd.

# 抽象类和纯虚函数

抽象类用于创建通用实体，就是您期待用其创建更具体的派生类。抽象类仅可以作为其他类的基类，这就是为什么不可能创建抽象类类型的对象。

一个至少包含一个纯虚函数的类就是抽象的。因此，源自于抽象类的类必须实现其全部的纯虚函数，否则它也将是抽象类。

虚函数通过使用pure-specifier语法来声明为"pure"。考虑CAnimal 类的示例，创建它只是为了提供普通函数 - CAnimal类型的对象对于实际使用过于一般。因此，CAnimal是抽象类的一个很好的示例：

```
class CAnimal
  {
public:
                     CAnimal();      // 构造函数
   virtual void      Sound() = 0;    // 纯虚函数
private:
   double            m_legs_count;   // 动物腿的数量
  };
```

这里Sound() 是一个纯虚函数，因为它是以纯虚函数PURE(=0)说明符进行声明。

纯虚函数只是以PURE说明符设置的虚函数：(=NULL) 或 (=0)。抽象类声明和使用的示例：

```
class CAnimal
  {
public:
   virtual void      Sound()=NULL;    // PURE 类函数，应该在派生类覆盖，CAnimal现在是抽象类 不
  };
//--- 派生自一个抽象类
class CCat : public CAnimal
 {
public:
  virtual void       Sound() { Print("Myau"); } // PURE被覆盖，CCat不是抽象类 可以创建
 };

//--- 错误使用的示例
new CAnimal;          // 'CAnimal'错误 - 编译器返回 "不能示例抽象类"的错误
CAnimal some_animal; // 'CAnimal'错误 - 编译器返回 "不能示例抽象类"的错误

//--- 正确使用的示例
new CCat;  // 无错误 - CCat 不是抽象类
CCat cat;  // 无错误 - CCat 不是抽象类
```

## 抽象类的限制

如果抽象类的构造函数调用一个纯虚函数 (直接或间接)，结果是未定义的。

```
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 抽象基类                                                         |
//+------------------------------------------------------------------+
class CAnimal
  {
public:
   //--- 纯虚函数
   virtual void       Sound(void)=NULL;
   //--- 函数
   void               CallSound(void) { Sound(); }
   //--- 构造函数
   CAnimal()
    {
     //--- 虚类函数的显式调用
     Sound();
     //--- 隐式调用（使用第三个函数）
     CallSound();
     //--- 构造函数和/或析构函数始终调用其自己的函数，
     //--- 即使它们是虚拟的并被派生类调用的函数覆盖
     //--- 如果调用的函数是纯虚函数，
     //--- 其调用将会导致危险的运行时错误："纯虚函数调用"
    }
  };
```

然而，对于抽象类的构造函数和析构函数可以调用其他成员函数。

© 2000-2025, MetaQuotes Ltd.

# 命名空间

namespace是一个特殊声明的区域，其中定义了各种ID：变量、函数、类等。其使用namespace关键字进行设置：

```
namespace name_of_space {
  //函数、类和变量定义的列表
}
```

应用'namespace'可以将全局的命名空间拆分为子空间。namespace中的所有ID都可以彼此使用，不需要特别指定。::操作符（语境解析操作）用于从外部访问命名空间（namespace）成员。

```
namespace ProjectData
{
class DataManager
  {
public:
   void              LoadData() {}
  };
void Func(DataManager& manager) {}
}
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 使用ProjectData命名空间
   ProjectData::DataManager mgr;
   mgr.LoadData();
   ProjectData::Func(mgr);
  }
```

命名空间（namespaces）用于以逻辑组的形式排列代码，并避免在程序中使用多个程序库时可能发生的名称冲突。在这种情况下，可以在其命名空间（namespaces）中声明每个程序库，以便显式访问每个程序库的必要函数和类。

命名空间（namespaces）可以在一个或多个文件中的多个块中声明。编译器在预处理期间将所有部分组合在一起，并且得到的命名空间包含所有部分中声明的所有成员。假设我们在Sample.mqh包含文件中实现了A类：

```
//+------------------------------------------------------------------+
//|                                                   Sample.mqh |
//+------------------------------------------------------------------+
class A
  {
public:
                     A() {Print(__FUNCTION__);}
  };
```

我们想在我们的项目中使用这个类，但是我们已经有A类。若要能够使用这两个类并要避免ID冲突，只需将包含文件包含在命名空间中：

© 2000-2025, MetaQuotes Ltd.

```
//--- 声明第一个A类
class A
  {
public:
                     A() {Print(__FUNCTION__);}
  };
//--- 将Sample.mqh文件中的A类包含在“程序库”命名空间中，以避免出现冲突
namespace Library
{
#include "Sample.mqh"
}
//--- 在“程序库”命名空间中添加另一个类
namespace Library
{
class B
  {
public:
                     B() {Print(__FUNCTION__);}
  };
}
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 使用全局命名空间中的A类
   A a1;
//--- 使用“程序库”命名空间中的A和B类
   Library::A a2;
   Library::B b;
  }
//+------------------------------------------------------------------+

/*
结果：
   A::A
   Library::A::A
   Library::B::B
*/
```

已经嵌套命名空间。嵌套命名空间可以无限制地访问其父系空间的成员，但是父系空间的成员不能无限制地访问嵌套命名空间。

```
namespace General
{
int Func();

namespace Details
{
```

© 2000-2025, MetaQuotes Ltd.

```
 int Counter;
 int Refresh()  {return Func(); }
}

int GetBars()   {return(iBars(Symbol(), Period()));};
int Size(int i) {return Details::Counter;}
}
```

## 全局命名空间(Globalnamespace)

如果ID没有在命名空间中显式声明，则认为它是全局命名空间的隐式部分。要显式地设置全局ID，请使用范围解析操作符，无需命名。这将可以使您将这个ID与位于不同命名空间中具有相同名称的任何其他元素区分开来。例如，导入函数时：

```
#import "lib.dll"
int Func();
#import
//+------------------------------------------------------------------+
//|  某个函数                                                        |
//+------------------------------------------------------------------+
int Func()
  {
   return(0);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//+--- 调用导入函数
   Print(lib::Func());
//+--- 调用我们的函数
   Print(::Func());
  }
```

在本例中，从DLL函数导入的所有函数都包含在相同名称的命名空间中。这使得编译器能够清楚地确定要调用的函数。

### 另见

全局变量、本地变量、可视范围和变量的生命周期、创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# 标准常量，列举和架构

为了简化编写程序并使其程序文本更加方便，在MQL5语言中预定义的标准常量和表达式，此外，结构服务也用于存储信息。

整型的标准变量与宏命令类似 。

此变量是按照用途分组的：

- 图表常量 与价格图表共同使用：开盘，导航，设置参量；
- 对象常量 是用来加工图表中添加和显示出来的图像对象；
- 指标常量 用于标准和自定义指标；
- 环境状态 描述MQL5程序的属性，显示客户端信息，安全性和活期存款账户；
- 交易常量 在交易过程中允许说明各类型信息；
- 命名常量 是MQL5语言中的常量；
- 数据结构 描述数据存储格式；
- 错误和警告代码 描述编译器信息和交易服务器发出的交易请求；
- 输入/输出常量 是用来与文件函数 一起工作，并在MessageBox()函数显示器上显示信息。

© 2000-2025, MetaQuotes Ltd.

# 图表常量

该常量描述了图表中个各种属性，并分成如下组：

- 事件类型 -与图表工作时时间发生；
- 图表时间表 -标准内置周期；
- 图表属性 -用作图表函数参量的标识符；
- 定位常量 -ChartNavigate()函数的参量值 ；
- 展示图表 - 设置图表外观.

© 2000-2025, MetaQuotes Ltd.

# 图表事件类型

使用预定函数OnChartEvent()能实现11种事件类型，自定义事件65535标识符在CHARTEVENT_CUSTOM 到 CHARTEVENT_CUSTOM_LAST 范围内浮动，为生成自定义事件，可以使用EventChartCustom()函数。

ENUM_CHART_EVENT

| ID | 说明 |
|---|---|
| CHARTEVENT_KEYDOWN | 打键次数 |
| CHARTEVENT_MOUSE_MOVE | 鼠标移动，鼠标点击 (如果CHART_EVENT_MOUSE_MOVE=true 为图表设置) |
| CHARTEVENT_MOUSE_WHEEL | 按下或滚动鼠标滚轮(如果CHART_EVENT_MOUSE_WHEEL=True为图表设置) |
| CHARTEVENT_OBJECT_CREATE | 图解对象 创建 (如果CHART_EVENT_OBJECT_CREATE=true 为图表设置) |
| CHARTEVENT_OBJECT_CHANGE | 图解对象 通过属性对话改变性质 |
| CHARTEVENT_OBJECT_DELETE | 图解对象 删除 (如果CHART_EVENT_OBJECT_DELETE=true 为图表设置) |
| CHARTEVENT_CLICK | 点击一个图表 |
| CHARTEVENT_OBJECT_CLICK | 点击 图解对象 |
| CHARTEVENT_OBJECT_DRAG | 拖放 图解对象 |
| CHARTEVENT_OBJECT_ENDEDIT | 在图表对象编辑里进行文本末尾编辑 |
| CHARTEVENT_CHART_CHANGE | 通过属性对话框改变图表大小或更改图表属性 |
| CHARTEVENT_CUSTOM | 事件的首写字符在自定义事件中 |
| CHARTEVENT_CUSTOM_LAST | 事件的最后一个字符在自定义事件中 |

对于每个事件类型，OnChartEvent()函数的输入参数都有固定值，这在此事件处理过程中是必须的，事件和值通过参量传递并列举在如下表格中

| 事件 | ID参量值 | lparam 参数值 | dparam 参数值 | sparam 参数值 |
|---|---|---|---|---|
| 按键事件 | CHARTEVENT_KEYDOWN | 按键代码 | 重复计数 (用户按住按键的重复击键次数) | 描述键盘按键状态的位掩码的字符串值 |
| 鼠标事件 (如果CHART_EVENT_ | CHARTEVENT_MOUSE_MOVE | X坐标 | Y坐标 | 描述鼠标按键状态的位掩码 |

© 2000-2025, MetaQuotes Ltd.

| 事件 | ID参量值 | lparam 参数值 | dparam 参数值 | sparam 参数值 |
|---|---|---|---|---|
| MOUSE_MOVE=true 为图表设置） | | | | 的字符串值 |
| 鼠标滚轮事件（如果 CHART_EVENT_MOUSE_WHEEL=true为图表设置） | CHARTEVENT_MOUSE_WHEEL | 键盘按键和鼠标按键的状态标识，鼠标指针的X和Y坐标。在以下示例查阅描述 | 鼠标滚轮滚动的Delta值 | — |
| 图解对象创建事件 （如果 CHART_EVENT_OBJECT_CREATE=true 为图表设置） | CHARTEVENT_OBJECT_CREATE | — | — | 创建的图表对象的名称 |
| 通过性质对话转变对象性质事件 | CHARTEVENT_OBJECT_CHANGE | — | — | 修改的图表对象的名称 |
| 图解对象删除事件 （如果 CHART_EVENT_OBJECT_DELETE=true为图表设置） | CHARTEVENT_OBJECT_DELETE | — | — | 删除图表对象的名称 |
| 鼠标点击图表事件 | CHARTEVENT_CLICK | X坐标 | Y坐标 | — |
| 鼠标点击属于图表的对象事件 | CHARTEVENT_OBJECT_CLICK | X坐标 | Y坐标 | 事件发生时图表对象的名称 |
| 使用鼠标拖动图解对象事件 | CHARTEVENT_OBJECT_DRAG | — | — | 移动图表对象名称 |
| 在编辑标签图表对象的进入访问完成文本编辑事件 | CHARTEVENT_OBJECT_ENDEDIT | — | — | 在文本编辑完成后编辑标签图表对象名称 |
| 在N数字下用户使用ID事件 | CHARTEVENT_CUSTOM+N | EventChartCustom() 函数的值 | EventChartCustom() 函数的值 | EventChartCustom() 函数的值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#define KEY_NUMPAD_5        12
#define KEY_LEFT            37
#define KEY_UP              38
#define KEY_RIGHT           39
#define KEY_DOWN            40
#define KEY_NUMLOCK_DOWN    98
#define KEY_NUMLOCK_LEFT   100
#define KEY_NUMLOCK_5      101
#define KEY_NUMLOCK_RIGHT  102
#define KEY_NUMLOCK_UP     104
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   Print("The expert with name ",MQL5InfoString(MQL5_PROGRAM_NAME)," is running");
//--- 启用对象创建事件
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- 启用对象删除事件
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//--- 强制更新图表属性确保做好事件处理的准备
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // 事件标识符
                  const long& lparam,   // 事件长整型参量
                  const double& dparam, // 事件双精度型参量
                  const string& sparam  // 事件字符串型参量
                  )
  {
//--- 鼠标左键点击图表
   if(id==CHARTEVENT_CLICK)
     {
      Print("The coordinates of the mouse click on the chart are: x = ",lparam,"  y =
     }
//--- 鼠标点击图形物件
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("The mouse has been clicked on the object with name '"+sparam+"'");
     }
//--- 按下密钥
   if(id==CHARTEVENT_KEYDOWN)
     {
      switch(lparam)
        {
         case KEY_NUMLOCK_LEFT:  Print("The KEY_NUMLOCK_LEFT has been pressed");   bre
         case KEY_LEFT:          Print("The KEY_LEFT has been pressed");           bre
         case KEY_NUMLOCK_UP:    Print("The KEY_NUMLOCK_UP has been pressed");     bre
         case KEY_UP:            Print("The KEY_UP has been pressed");             bre
         case KEY_NUMLOCK_RIGHT: Print("The KEY_NUMLOCK_RIGHT has been pressed"); bre
         case KEY_RIGHT:         Print("The KEY_RIGHT has been pressed");          bre
         case KEY_NUMLOCK_DOWN:  Print("The KEY_NUMLOCK_DOWN has been pressed");   bre
         case KEY_DOWN:          Print("The KEY_DOWN has been pressed");           bre
         case KEY_NUMPAD_5:      Print("The KEY_NUMPAD_5 has been pressed");       bre
         case KEY_NUMLOCK_5:     Print("The KEY_NUMLOCK_5 has been pressed");      bre
         default:                Print("Some not listed key has been pressed");
        }
      ChartRedraw();
     }
//--- 物件被删除了
   if(id==CHARTEVENT_OBJECT_DELETE)
     {
      Print("The object with name ",sparam," has been deleted");
     }
//--- 物件被创建了
   if(id==CHARTEVENT_OBJECT_CREATE)
     {
      Print("The object with name ",sparam," has been created");
     }
//--- 移动物件或者更改定位点坐标
   if(id==CHARTEVENT_OBJECT_DRAG)
     {
      Print("The anchor point coordinates of the object with name ",sparam," has been
     }
//--- 物件编辑文本被更改
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
     {
      Print("The text in the Edit field of the object with name ",sparam," has been ch
```

© 2000-2025, MetaQuotes Ltd.

```
      }
  }
```

对于CHARTEVENT_MOUSE_MOVE事件 **sparam** 字符串参数包含有关键盘和鼠标按钮的状态信息：

| Bit | 描述 |
|---|---|
| 1 | 鼠标左按键状态 |
| 2 | 鼠标右按键状态 |
| 3 | SHIFT 按钮状态 |
| 4 | CTRL 按钮状态 |
| 5 | 鼠标中按键状态 |
| 6 | 第一个额外鼠标键的状态 |
| 7 | 第二个额外鼠标键的状态 |

示例：

```
//+------------------------------------------------------------------+
//| EA初始化函数                                                     |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 启用 CHART_EVENT_MOUSE_MOVE 信息
   ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
//--- 禁用图表的上下文菜单 (在右侧)
   ChartSetInteger(0,CHART_CONTEXT_MENU,0);
//--- 禁用十字准线 (通过中间按钮)
   ChartSetInteger(0,CHART_CROSSHAIR_TOOL,0);
//--- 强制更新图表属性确保做好事件处理的准备
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 鼠标点选位置                                                     |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // 鼠标左按键
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // 鼠标右按键
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // 鼠标中按键
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // 鼠标第一个 X 键
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // 鼠标第二个 X 键
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // shift 键
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // control 键
   return(res);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
  {
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
  }
```

对于CHARTEVENT_MOUSE_WHEEL事件，参数**lparam** 和**dparam** 包含**Ctrl**和 **Shift**键，鼠标按键，光标坐标的状态信息和鼠标滚轮滚动值。为了更好的理解，请在图表上运行这个EA交易并滚动鼠标滚轮，同时按下不同的按键并按住代码中所描述的按键。

CHARTEVENT_MOUSE_WHEEL 事件处理示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
init OnInit()
  {
//--- 启用鼠标滚轮来滚动信息
   ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
//--- 强制更新图表属性确保做好事件处理的准备
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
  {
   if(id==CHARTEVENT_MOUSE_WHEEL)
     {
      //--- 考虑这个事件的鼠标按键和滚轮的状态
      int flg_keys = (int)(lparam>>32);          // Ctrl 和 Shift 键，鼠标按键的状态标识
      int x_cursor = (int)(short)lparam;         // 鼠标滚轮事件发生的X坐标
      int y_cursor = (int)(short)(lparam>>16);   // 鼠标滚轮事件发生的Y坐标
      int delta    = (int)dparam;                // 鼠标滚动的总值，到达 +120 或 -120 时触
      //--- 处理标识
      string str_keys="";
      if((flg_keys&0x0001)!=0) str_keys+="LMOUSE ";
      if((flg_keys&0x0002)!=0) str_keys+="RMOUSE ";
      if((flg_keys&0x0004)!=0) str_keys+="SHIFT ";
      if((flg_keys&0x0008)!=0) str_keys+="CTRL ";
      if((flg_keys&0x0010)!=0) str_keys+="MMOUSE ";
      if((flg_keys&0x0020)!=0) str_keys+="X1MOUSE ";
      if((flg_keys&0x0040)!=0) str_keys+="X2MOUSE ";
```

© 2000-2025, MetaQuotes Ltd.

```
      if(str_keys!="")
         str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1) + "'";
      PrintFormat("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x,
     }
  }
//+------------------------------------------------------------------+ /*
   Example of the output
   CHARTEVENT_MOUSE_WHEEL: Ctrl pressed: X=193, Y=445, delta=-120
   CHARTEVENT_MOUSE_WHEEL: Shift pressed: X=186, Y=446, delta=120
   CHARTEVENT_MOUSE_WHEEL:  X=178, Y=447, delta=-120
   CHARTEVENT_MOUSE_WHEEL:  X=231, Y=449, delta=120
   CHARTEVENT_MOUSE_WHEEL: MiddleButton pressed: X=231, Y=449, delta=120
   CHARTEVENT_MOUSE_WHEEL: LeftButton pressed: X=279, Y=320, delta=-120
   CHARTEVENT_MOUSE_WHEEL: RightButton pressed: X=253, Y=330, delta=120  */
```

另见

事件处理函数，工作事件

© 2000-2025, MetaQuotes Ltd.

# 图表时间表

所有预定义的图表时间表都有唯一的标识符，当MQL5程序运行时，PERIOD_CURRENT 标识符代表图表本期。

ENUM_TIMEFRAMES

| ID | 描述 |
|---|---|
| PERIOD_CURRENT | 当前时间表 |
| PERIOD_M1 | 1 分钟 |
| PERIOD_M2 | 2 分钟 |
| PERIOD_M3 | 3 分钟 |
| PERIOD_M4 | 4 分钟 |
| PERIOD_M5 | 5 分钟 |
| PERIOD_M6 | 6 分钟 |
| PERIOD_M10 | 10 分钟 |
| PERIOD_M12 | 12 分钟 |
| PERIOD_M15 | 15 分钟 |
| PERIOD_M20 | 20 分钟 |
| PERIOD_M30 | 30 分钟 |
| PERIOD_H1 | 1小时 |
| PERIOD_H2 | 2 小时 |
| PERIOD_H3 | 3 小时 |
| PERIOD_H4 | 4 小时 |
| PERIOD_H6 | 6小时 |
| PERIOD_H8 | 8小时 |
| PERIOD_H12 | 12 小时 |
| PERIOD_D1 | 1 天 |
| PERIOD_W1 | 1 周 |
| PERIOD_MN1 | 1月 |

示例：

```
   string chart_name="test_Object_Chart";
   Print("Let's try to create a Chart object with the name ",chart_name);
//--- 如果没有这个物件-创建它
   if(ObjectFind(0,chart_name)<0)ObjectCreate(0,chart_name,OBJ_CHART,0,0,0,0,0);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义交易品种
   ObjectSetString(0,chart_name,OBJPROP_SYMBOL,"EURUSD");
//--- 设置定位点X坐标
   ObjectSetInteger(0,chart_name,OBJPROP_XDISTANCE,100);
//--- 设置定位点Y坐标
   ObjectSetInteger(0,chart_name,OBJPROP_YDISTANCE,100);
//--- 设置图表宽度
   ObjectSetInteger(0,chart_name,OBJPROP_XSIZE,400);
//--- 设置高度
   ObjectSetInteger(0,chart_name,OBJPROP_YSIZE,300);
//--- 设置时间表
   ObjectSetInteger(0,chart_name,OBJPROP_PERIOD,PERIOD_D1);
//--- 设置规模(从 0 到 5)
   ObjectSetDouble(0,chart_name,OBJPROP_SCALE,4);
//--- 禁止鼠标选择
   ObjectSetInteger(0,chart_name,OBJPROP_SELECTABLE,false);
```

## Timeseries identifiers

时间序列标识符使用在iHighest()和iLowest()函数中。它们可以等于一个枚举值

**ENUM_SERIESMODE**

| 标识符 | 描述 |
|---|---|
| MODE_OPEN | 开盘价 |
| MODE_LOW | 最低价 |
| MODE_HIGH | 最高价 |
| MODE_CLOSE | 收盘价 |
| MODE_VOLUME | 报价量 |
| MODE_REAL_VOLUME | 真实交易量 |
| MODE_SPREAD | 点差 |

另见

秒周期，周期，日期和时间，对象可见性

© 2000-2025, MetaQuotes Ltd.

# 图表属性

ENUM_CHART_PROPERTY 计数标识符与使用工作图表函数参量相同。在类型属性列里的缩写r/o表示只读，不能更改，当访问某一功能时，指定额外参数修饰语是有必要的，该修饰语提供了图表窗口的数量，0是主窗口。 "属性类型"栏中的w/o缩写意味着该属性只能写入，不能接收。当访问某个属性时，必须指定额外的参数修饰符，这用来显示图表子窗口的数量。 0 意味着主窗口。

定义图表属性的函数事实上就是用来将变更命令发送给图表。如果这些函数成功执行，常见的图表事件队列就会包含该命令。当处理图表事件队列时，这些更改就会实施到图表。

因此，不要期待调用这些函数后，图表马上的视觉升级。通常，图表会在更改事件后，通过程序端自动升级 - 有新报价进来，重新调整窗口大小等等。使用ChartRedraw() 函数以便有力地更新图表。

函数ChartSetInteger()和ChartGetInteger() 。

**ENUM_CHART_PROPERTY_INTEGER**

| ID | 描述 | 性质类型 |
|---|---|---|
| CHART_SHOW | 绘制价格图表。false情况下，不可绘制价格图表的任何属性并删除首行图表边界，包括时间和价格范围，快速导航条，日历事件标签，交易标签，指标和柱形图提示，指标子窗口，交易量直方图等等。禁用绘制是使用图形资源创建自定义程序界面的理想方案。无论CHART_SHOW属性值多少，都会始终绘制图形对象。 | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_IS_OBJECT | 识别"图表"(OBJ_CHART) 对象 - 图形对象返回true。真实图表返回false | bool r/o |
| CHART_BRING_TO_TOP | 显示其他图表之上的图表 | bool |
| CHART_CONTEXT_MENU | 启用/禁用右击访问快捷菜单。当CHART_CONTEXT_MENU=false时，仅禁用图表快捷菜单。图表上对象的快捷菜单仍可使用。 | bool 默认为true） |
| CHART_CROSSHAIR_TOOL | 启用/禁用中间单击访问十字光标工具。 | bool 默认为true） |
| CHART_MOUSE_SCROLL | 使用鼠标左键水平滚动图表。如果以下属性的值设为true也可以垂直滚动：CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR 当CHART_MOUSE_SCROLL=false时，不可使用鼠标 | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| | 滚轮滚动图表。 | |
| CHART_EVENT_MOUSE_WHEEL | 发送鼠标滚轮事件的消息 (CHARTEVENT_MOUSE_WHEEL) 到图表上的所有mq5程序 | bool 默认为true） |
| CHART_EVENT_MOUSE_MOVE | 发送鼠标移动和鼠标点击事件的通知 (CHARTEVENT_MOUSE_MOVE) 到图表上的所有mq5程序 | bool |
| CHART_EVENT_OBJECT_CREATE | 发送创建新对象的事件通知 (CHARTEVENT_OBJECT_CREATE) 到图表上的所有mq5程序 | bool |
| CHART_EVENT_OBJECT_DELETE | 发送删除对象的事件通知 (CHARTEVENT_OBJECT_DELETE) 到图表上的所有mq5程序 | bool |
| CHART_MODE | 图表类型（蜡烛台、字节或线） | enum ENUM_CHART_MODE |
| CHART_FOREGROUND | 背景的价格图表 | bool |
| CHART_SHIFT | 左边缩进价格图表 | bool |
| CHART_AUTOSCROLL | 自动移向图表的右边框 | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_KEYBOARD_CONTROL | 可以使用键盘管理图表（"Home"，"End"，"PageUp"，"+"，"-"，"up arrow"等）。设置为false的CHART_KEYBOARD_CONTROL在保留接收按下OnChartEvent()函数事件按键的能力时，禁止图表滚动和缩放。 | bool |
| CHART_QUICK_NAVIGATION | 允许图表拦截空格键和回车键来激活快速导航栏。双击鼠标或按下空格/回车键后，快速导航栏自动出现在图表底部。这可以使您快速更改交易品种，时间周期和第一可视栏日期。 | bool |
| CHART_SCALE | 测量 | int 从0到5 |
| CHART_SCALEFIX | 固定标盘模式 | bool |
| CHART_SCALEFIX_11 | 测量方式1:1 | bool |
| CHART_SCALE_PT_PER_BAR | 每字节指定相关测量 | bool |
| CHART_SHOW_TICKER | 在左上角显示交易品种报价行情。 | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| | 将CHART_SHOW_TICKER设置为'false'也会将CHART_SHOW_OHLC设置为'false'并禁用OHLC | |
| CHART_SHOW_OHLC | 在左上角显示OHLC值。将CHART_SHOW_OHLC设置为'true'也会将CHART_SHOW_TICKER设置为'true'并启用报价行情 | bool |
| CHART_SHOW_BID_LINE | 在图表水平线上显示出价值 | bool |
| CHART_SHOW_ASK_LINE | 在图表水平线上显示要价值 | bool |
| CHART_SHOW_LAST_LINE | 在图表水平线上显示最终值 | bool |
| CHART_SHOW_PERIOD_SEP | 在相邻周期显示垂直分离器 | bool |
| CHART_SHOW_GRID | 在图表中显示网格 | bool |
| CHART_SHOW_VOLUMES | 在图表中显示成交量 | enum ENUM_CHART_VOLUME_MODE |
| CHART_SHOW_OBJECT_DESCR | 显示对象的文本描述(不可用于所有对象) | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_SHOW_TRADE_HISTORY | 在图表上以进/出箭头的形式显示交易历史中的交易。参见平台设置中“显示交易历史”选项说明。 | bool |
| CHART_VISIBLE_BARS | 图表上显示的字节数量 | int r/o |
| CHART_WINDOWS_TOTAL | 图表窗口总数，包括指标预览窗口 | int r/o |
| CHART_WINDOW_IS_VISIBLE | 预览窗口可见性 | boolr/o 修饰符 -子窗口号 |
| CHART_WINDOW_HANDLE | 处理图表窗口 (HWND) | int r/o |
| CHART_WINDOW_YDISTANCE | 指标子窗口的上帧与主图表窗口的上帧之间的距离，沿着Y轴垂直，以像素为单位。在鼠标事件情况下，根据图表主窗口的坐标传递光标坐标，而将指标子窗口的图形对象坐标设置相对于子窗口的左上角。<br>需要该值是为了将主图表的绝对坐标转换为子窗口的本地坐标，以便图形对象的正确工作，该坐标设置 | int r/o 修饰符 -子窗口号 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| | 位置相对于子窗口帧的左上角。 | |
| CHART_FIRST_VISIBLE_BAR | 图表中第一可见字节字符。字节索引同于时序列。 | int r/o |
| CHART_WIDTH_IN_BARS | 以字节转发图表 | int r/o |
| CHART_WIDTH_IN_PIXELS | 像素转发图表 | int r/o |
| CHART_HEIGHT_IN_PIXELS | 图表像素高度 | int 修饰符 - 子窗口号 |
| CHART_COLOR_BACKGROUND | 图表背景颜色 | color |
| CHART_COLOR_FOREGROUND | 轴线、缩放和OHLC线的颜色 | color |
| CHART_COLOR_GRID | 网格颜色 | color |
| CHART_COLOR_VOLUME | 成交量颜色和开仓水平 | color |
| CHART_COLOR_CHART_UP | 上升字节、阴影和大型烛台整体边界的颜色 | color |
| CHART_COLOR_CHART_DOWN | 下降字节、阴影和支撑烛台的颜色 | color |
| CHART_COLOR_CHART_LINE | 折线图颜色和日语烛台躲闪颜色 | color |
| CHART_COLOR_CANDLE_BULL | 大型烛台主体颜色 | color |
| CHART_COLOR_CANDLE_BEAR | 承受烛台主体颜色 | color |
| CHART_COLOR_BID | 出价水平颜色 | color |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_COLOR_ASK | 要价水平颜色 | color |
| CHART_COLOR_LAST | 最后执行交易价格水平线颜色（Last） | color |
| CHART_COLOR_STOP_LEVEL | 停止订购水平颜色（斩仓和获利） | color |
| CHART_SHOW_TRADE_LEVELS | 在图表中显示交易水平（开仓水平、斩仓、获利和代办订单） | bool |
| CHART_DRAG_TRADE_LEVELS | 用鼠标在图表上拖拽交易水平的权限。拖拽模式默认启用（true 值） | bool |
| CHART_SHOW_DATE_SCALE | 显示图表的时间比例 | bool |
| CHART_SHOW_PRICE_SCALE | 显示图表的价格比例 | bool |
| CHART_SHOW_ONE_CLICK | 显示 "单击交易"面板在图表上 | bool |
| CHART_IS_MAXIMIZED | 图表窗口最大化 | bool r/o |
| CHART_IS_MINIMIZED | 图表窗口最小化 | bool r/o |
| CHART_IS_DOCKED | 图表窗口靠边。如果设置为false，图表可能被拖拽到程序端区域以外 | bool |
| CHART_FLOAT_LEFT | 相对于虚拟屏幕的没有靠边的图表 | int |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| | 窗口的左坐标 | |
| CHART_FLOAT_TOP | 相对于虚拟屏幕的没有靠边的图表窗口的顶部坐标 | int |
| CHART_FLOAT_RIGHT | 相对于虚拟屏幕的没有靠边的图表窗口的右坐标 | int |
| CHART_FLOAT_BOTTOM | 相对于虚拟屏幕的没有靠边的图表窗口的底部坐标 | int |

函数 ChartSetDouble() 和 ChartGetDouble()

ENUM_CHART_PROPERTY_DOUBLE

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_SHIFT_SIZE | 大小在右边百分比边界从零字节缩进 | double (10%-50%) |
| CHART_FIXED_POSITION | 图表以百分比定位左边界的位置。在水平时间轴用小的灰白三角形标记定位的图表。只有在禁止进入订单号时图表自动滚到右侧它才会显示(见 CHART_AUTOSCROLL property)。放大缩小时，定位的柱保留原位。 | 双精度 |
| CHART_FIXED_MAX | 固定图表最大值 | 双精度 |
| CHART_FIXED_MIN | 固定图表最小值 | 双精度 |
| CHART_POINTS_PER_BAR | 测量相关的每个字节 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_PRICE_MIN | 图表最小值 | double r/o 修饰符 - 子窗口号 |
| CHART_PRICE_MAX | 图表最大值 | double r/o 修饰符 - 子窗口号 |

ChartSetString() 和 ChartGetString() 函数

ENUM_CHART_PROPERTY_STRING

| ID | 描述 | 性质类型 |
| --- | --- | --- |
| CHART_COMMENT | 图表中的评论文本 | string |
| CHART_EXPERT_NAME | 在指定chart_id图表上运行的EA交易的名称 | string r/o |
| CHART_SCRIPT_NAME | 在指定chart_id图表上运行的脚本名称 | string r/o |

示例：

```
int chartMode=ChartGetInteger(0,CHART_MODE);
switch(chartMode)
  {
   case(CHART_BARS):    Print("CHART_BARS");   break;
   case(CHART_CANDLES): Print("CHART_CANDLES");break;
   default:Print("CHART_LINE");
  }
bool shifted=ChartGetInteger(0,CHART_SHIFT);
if(shifted) Print("CHART_SHIFT = true");
else Print("CHART_SHIFT = false");
bool autoscroll=ChartGetInteger(0,CHART_AUTOSCROLL);
if(autoscroll) Print("CHART_AUTOSCROLL = true");
else Print("CHART_AUTOSCROLL = false");
int chartHandle=ChartGetInteger(0,CHART_WINDOW_HANDLE);
Print("CHART_WINDOW_HANDLE = ",chartHandle);
int windows=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
Print("CHART_WINDOWS_TOTAL = ",windows);
if(windows>1)
  {
   for(int i=0;i<windows;i++)
     {
      int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,i);
      double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,i);
      double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,i);
      Print(i+": CHART_HEIGHT_IN_PIXELS = ",height," pixels");
      Print(i+": CHART_PRICE_MIN = ",priceMin);
      Print(i+": CHART_PRICE_MAX = ",priceMax);
```

© 2000-2025, MetaQuotes Ltd.

```
            }
        }
```

另见

使用图表示例

© 2000-2025, MetaQuotes Ltd.

# 定位常量

ENUM_CHART_POSITION 列表中的三个标识符有可能是ChartNavigate()函数的方位参量的值。

**ENUM_CHART_POSITION**

| ID | 描述 |
|---|---|
| CHART_BEGIN | 图表开始（最原始价位） |
| CHART_CURRENT_POS | 当前位置 |
| CHART_END | 图表末尾（当前价格） |

示例：

```
long handle=ChartOpen("EURUSD",PERIOD_H12);
if(handle!=0)
  {
   ChartSetInteger(handle,CHART_AUTOSCROLL,false);
   ChartSetInteger(handle,CHART_SHIFT,true);
   ChartSetInteger(handle,CHART_MODE,CHART_LINE);
   ResetLastError();
   bool res=ChartNavigate(handle,CHART_END,150);
   if(!res) Print("Navigate failed. Error = ",GetLastError());
   ChartRedraw();
  }
```

© 2000-2025, MetaQuotes Ltd.

# 图表陈述

价格图表可以以下三种方式陈列：

- 字符；
- 蜡烛台；
- 反射线。

显示价格图表的特殊方式是通过ChartSetInteger(chart_handle,CHART_MODE,chart_mode),函数建立，chart_mode是ENUM_CHART_MODE计算式中的一个值。

**ENUM_CHART_MODE**

| ID | 描述 |
|---|---|
| CHART_BARS | 以序列字符陈列 |
| CHART_CANDLES | 以日语蜡烛台陈列 |
| CHART_LINE | 收盘价格线型显示 |

在价格表格上显示成交量的特殊方式是通过使用ChartSetInteger(chart_handle,CHART_SHOW_VOLUMES,volume_mode)函数，volume_mode是计算式ENUM_CHART_VOLUME_MODE中值的一个。

**ENUM_CHART_VOLUME_MODE**

| ID | 描述 |
|---|---|
| CHART_VOLUME_HIDE | 成交量隐藏 |
| CHART_VOLUME_TICK | 最小价格成交量 |
| CHART_VOLUME_REAL | 交易成交量 |

示例：

```
//--- 获得当前图表处理权
   long handle=ChartID();
   if(handle>0) // 如果成功，加上自定义
     {
      //--- 禁止自动滚动
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- 设置图表右缩进
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- 显示蜡烛图
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- 从历史记录起始位置按100柱为一页滚动
      ChartNavigate(handle,CHART_CURRENT_POS,100);
      //--- 设置订单交易量显示模式
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
     }
```

© 2000-2025, MetaQuotes Ltd.

另见：

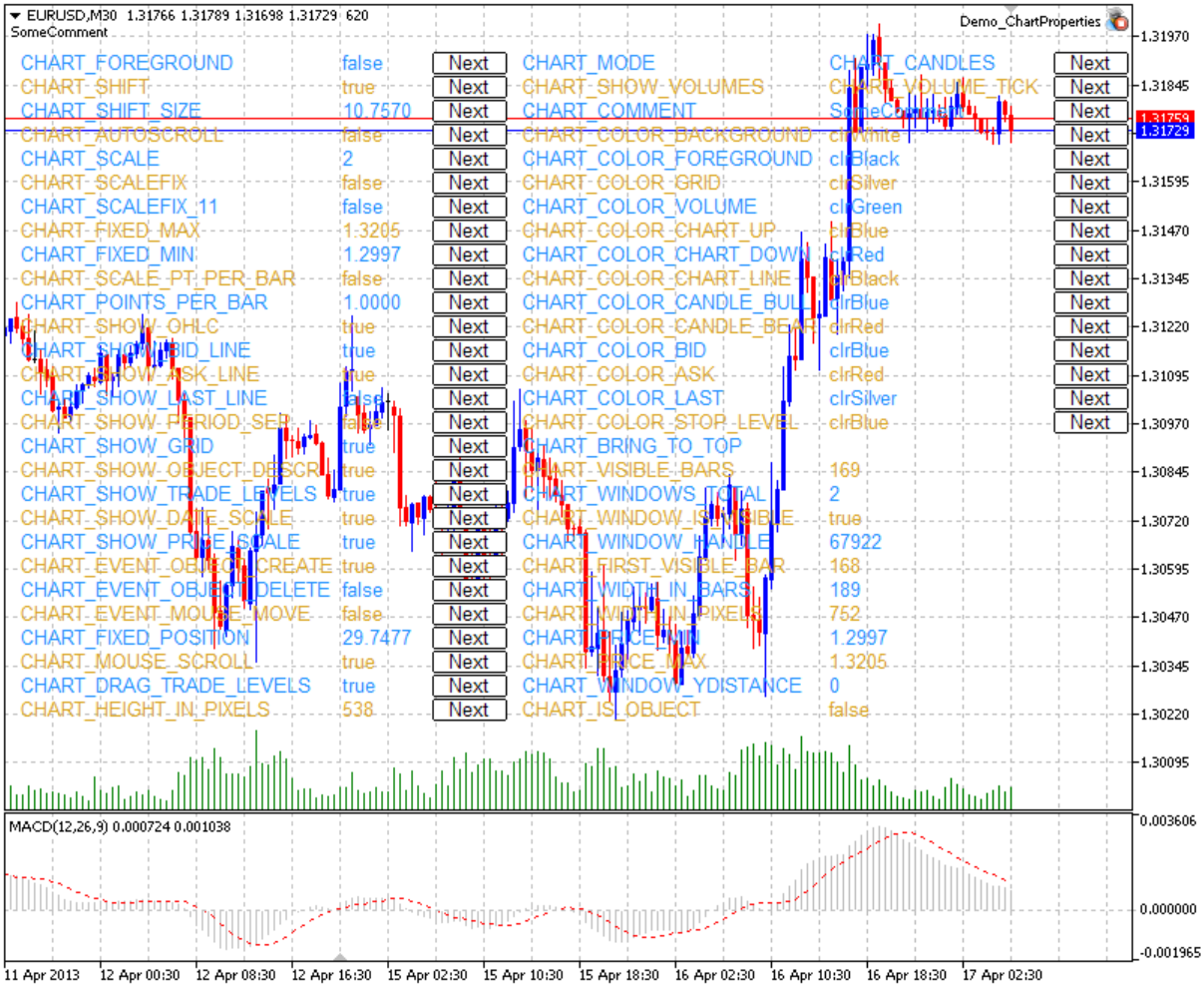
打开图表，图表ID

© 2000-2025, MetaQuotes Ltd.

# 使用图表示例

本节包含图表属性工作的示例。每个属性显示一到两个完整的函数。这些函数允许设置/接收属性的值。这些函数在自定义mql5应用程序中可以被使用"as is"。

下面的截图举例说明地展示了如何改变图表属性表现的图形面板。点击下一步可以设置新的适当属性的值和查看图表窗口的变化。




面板的源代码位于 下面。

## 图表属性和使用它们的样本函数

- **CHART_IS_OBJECT** 定义对象是否是真实图表或是 图形对象。

```
//+------------------------------------------------------------------+
//| 定义对象是否是图表。如果它是                                     |
//| 图形对象，则结果为true。如果它是真实                             |
//| 图表，则结果变量为false值。                                      |
//+------------------------------------------------------------------+
bool ChartIsObject(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 获得图表属性
   if(!ChartGetInteger(chart_ID,CHART_IS_OBJECT,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      //--- 返回 false
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
```

- **CHART_BRING_TO_TOP** 显示所有其他图表的顶部的图表。

```
//+------------------------------------------------------------------+
//| 发送命令到程序端展示所有其他图表上面的图表。                                  |
//+------------------------------------------------------------------+
bool ChartBringToTop(const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 显示所有其他图表的顶部的图表
   if(!ChartSetInteger(chart_ID,CHART_BRING_TO_TOP,0,true))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_MOUSE_SCROLL** 使用鼠标左键滚动图表的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义了使用鼠标左键滚动图表是否                                        |
//| 被启用。                                                          |
//+------------------------------------------------------------------+
bool ChartMouseScrollGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用使用鼠标左键滚动图表。                                |
//+------------------------------------------------------------------+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_EVENT_MOUSE_MOVE** 是发送有关移动事件和鼠标点击mql5应用程序(CHARTEVENT_MOUSE_MOVE)的信息的属性。

```
//+------------------------------------------------------------------+
//| 检查有关移动事件和鼠标点击的信息                                    |
//| 被发送到图表上的所有mql5应用程序                                    |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 该函数启用/禁用发送信息模式，有关移动                                          |
//| 事件和在图表上鼠标点击mql5应用程序。                                           |
//+------------------------------------------------------------------------------+
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_EVENT_OBJECT_CREATE** 是发送有关创建图形对象事件到mql5应用程序(CHARTEVENT_OBJECT_CREATE)的信息的属性。

```
//+------------------------------------------------------------------+
//| 检查是否有关创建图形对象事件的信息                                 |
//| 被发送到图表上的所有mql5应用程序                                   |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 该函数启用/禁用发送信息模式，有关                                              |
//| 图形对象创建事件到图表上所有的mql5应用程序。                                   |
//+------------------------------------------------------------------------------+
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_EVENT_OBJECT_DELETE** 是发送有关图表对象删除事件到mql5应用程序(CHARTEVENT_OBJECT_DELETE)的信息的属性。

```
//+------------------------------------------------------------------+
//| 检查有关图形对象删除事件的信息                                     |
//| 被发送到图表上的所有mql5应用程序                                   |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用发送信息模式，有关                                |
//| 发送至所有mql5应用程序的图形对象删除事件的信息。                 |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_MODE** - 图表类型（蜡烛图，柱形图或线型图）。

```
//+------------------------------------------------------------------+
//| 获得图表展示类型（蜡烛图，柱形图或                               |
//| 线型图）。                                                       |
//+------------------------------------------------------------------+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=WRONG_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((ENUM_CHART_MODE)result);
  }
//+------------------------------------------------------------------+
//| 设置图表显示类型（蜡烛图，柱形图或                               |
//| 线型图）。                                                       |
//+------------------------------------------------------------------+
bool ChartModeSet(const long value,const long chart_ID=0)
  {
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_MODE,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_FOREGROUND**在前台显示价格图表的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义是否价格图表显示在                                     |
//| 前台。                                                           |
//+------------------------------------------------------------------+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 函数启用/禁用价格图表展示模式在                                  |
//| 前台。                                                           |
//+------------------------------------------------------------------+
bool ChartForegroundSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- 在EA日志显示错误信息
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHIFT** - 从右边缘移动价格图表的模式。

```
//+------------------------------------------------------------------------------------------+
//| 函数定义是否从右边缘移动价格图表的模式
//| 被启用。
//+------------------------------------------------------------------------------------------+
bool ChartShiftGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 函数启用/禁用价格图表展示模式                                                |
//| 从右边缘开始移动。                                                           |
//+------------------------------------------------------------------------------+
bool ChartShiftSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
```

- **CHART_AUTOSCROLL** - 自动移动到图表右边缘的模式。

```
//+------------------------------------------------------------------+
//| 函数定义了新报价进入时是否启用                                   |
//| 图表向右的自动滚动模式。                                         |
//+------------------------------------------------------------------+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用新报价进入时，图表向右的                          |
//| 自动滚动模式。                                                   |
//+------------------------------------------------------------------+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SCALE** - 图表比例属性。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 获得图表比例 从0到5)。                                           |
//+------------------------------------------------------------------+
int ChartScaleGet(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
//+------------------------------------------------------------------+
//| 设置图表比例 从0到5)。                                           |
//+------------------------------------------------------------------+
bool ChartScaleSet(const long value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SCALEFIX** - 固定图表比例模式。

```
//+------------------------------------------------------------------+
//| 函数定义了是否启用固定比例模式。                                 |
//+------------------------------------------------------------------+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 函数启用/禁用固定比例模式。                                      |
//+------------------------------------------------------------------+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SCALEFIX_11** - 1:1 图表比例模式。

```
//+------------------------------------------------------------------+
//| 该函数定义了是否启用“1:1”比例。                                  |
//+------------------------------------------------------------------+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
```

© 2000-2025, MetaQuotes Ltd.

```
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 函数启用/禁用 "1:1"比例模式                                      |
//+------------------------------------------------------------------+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SCALE_PT_PER_BAR** - 指定每柱图表比例点数。

```
//+------------------------------------------------------------------+
//| 函数定义了是否启用指定每柱图表比例点数的                         |
//| 模式。                                                           |
//+------------------------------------------------------------------+
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 函数启用/禁用指定每柱图表比例点数的                              |
```

© 2000-2025, MetaQuotes Ltd.

```
//| 模式。
//+------------------------------------------------------------------------------------+
bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_OHLC** - 在左上角展示OHLC值的属性。

```
//+------------------------------------------------------------------+
//| 函数定义了是否启用在左上角展示OHLC值                                |
//| 的模式。                                                          |
//+------------------------------------------------------------------+
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------+
//| 该函数启用/禁用在图表左上角显示OHLC值                                    |
//| 的模式。                                                               |
//+------------------------------------------------------------------------+
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_BID_LINE** - 图表水平线显示卖价值的属性。

```
//+------------------------------------------------------------------------------+
//| 函数定义了在图表展示卖价值的模式是否                                          |
//| 被启用。                                                                     |
//+------------------------------------------------------------------------------+
bool ChartShowBidLineGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用在图表上展示卖价行的模式。                        |
//+------------------------------------------------------------------+
bool ChartShowBidLineSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_ASK_LINE** - 图表水平线显示买价值的属性。

```
//+------------------------------------------------------------------+
//| 函数定义了是否启用在图表上展示买价值线的模式。                   |
//+------------------------------------------------------------------+
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用在图表上显示买价行的模式。                        |
//+------------------------------------------------------------------+
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

- **CHART_SHOW_LAST_LINE** -图表水平线显示最近值的属性。

```
//+------------------------------------------------------------------------------------+
//| 该函数定义了是否启用显示最近执行的交易的价格行模式。                                |
//+------------------------------------------------------------------------------------+
bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------------+
//| 函数启用/禁用显示最近执行的交易的价格行模式。                                      |
//+------------------------------------------------------------------------------------+
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_PERIOD_SEP** -在相邻周期间展示垂直分隔符的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义了是否启用在相邻周期间                                 |
//| 展示垂直分隔符的模式。                                           |
//+------------------------------------------------------------------+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//|该函数启用/禁用在相邻周期间                                        |
//|  展示垂直分隔符的模式。                                           |
//+------------------------------------------------------------------+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_GRID** - 展示图表网格的属性。

```
//+------------------------------------------------------------------+
//|  该函数定义了是否展示图表网格。                                   |
//+------------------------------------------------------------------+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
```

```
   if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用图表网格。                                          |
//+------------------------------------------------------------------+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_VOLUMES** - 在图表上展示交易量的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义了是否在图表上展示交易量 (不                             |
//| 显示，显示订单交易量，显示实际交易量)。                            |
//+------------------------------------------------------------------+
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=WRONG_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   return((ENUM_CHART_VOLUME_MODE)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置在图表上显示交易量的模式。                                |
//+------------------------------------------------------------------+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_VOLUMES,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_OBJECT_DESCR** - 图形对象弹出描述的属性。

```
//+------------------------------------------------------------------+
//| 函数定义了悬停鼠标时是否显示                                       |
//| 图形对象的弹出描述 。                                             |
//+------------------------------------------------------------------+
bool ChartShowObjectDescriptionGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用鼠标悬停时                                         |
//| 显示图形对象弹出描述的模式。                                       |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool ChartShowObjectDescriptionSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_VISIBLE_BARS** 定义图表上可用于展示的柱的数量。

```
//+------------------------------------------------------------------+
//| 函数接收图表窗口中                                               |
//| 显示的柱数 可见)。                                               |
//+------------------------------------------------------------------+
int ChartVisibleBars(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_WINDOWS_TOTAL** 定义包括指标子窗口的图表窗口总数。

```
//+------------------------------------------------------------------+
//| 该函数获得所有图表窗口的总数包括指标                             |
//| 子窗口。                                                         |
//+------------------------------------------------------------------+
int ChartWindowsTotal(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_WINDOW_IS_VISIBLE** 定义子窗口的可视性。

```
//+------------------------------------------------------------------+
//| 该函数定义了当前图表窗口或子窗口是否                                |
//| 可见。                                                           |
//+------------------------------------------------------------------+
bool ChartWindowsIsVisible(bool &result,const long chart_ID=0,const int sub_window=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_IS_VISIBLE,sub_window,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
```

- **CHART_WINDOW_HANDLE** 返回图表句柄。

```
//+------------------------------------------------------------------+
//| 函数获取图表句柄                                                 |
//+------------------------------------------------------------------+
int ChartWindowsHandle(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_HANDLE,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_WINDOW_YDISTANCE** 定义指标子窗口上边框和图表主窗口上边框之间的像素距离。

```
//+------------------------------------------------------------------+
//| 该函数获得子窗口上边框和                                         |
//| 图表主窗口上边框之间的像素距离。                                 |
//+------------------------------------------------------------------+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_FIRST_VISIBLE_BAR** 返回图表上最先可见的柱的数量（与 时间序列 相关的柱）。

```
//+------------------------------------------------------------------+
//| 该函数接收图表上最先可见的柱数。                                 |
//| 像在时间序列一样执行索引，最近的柱指数更小。                     |
//+------------------------------------------------------------------+
int ChartFirstVisibleBar(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_FIRST_VISIBLE_BAR,0,result))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 在EA日志显示错误信息
       Print(__FUNCTION__+", Error Code = ",GetLastError());
      }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_WIDTH_IN_BARS** 返回图表柱形宽度。

```
//+------------------------------------------------------------------+
//| 函数接收图表柱形宽度。                                           |
//+------------------------------------------------------------------+
int ChartWidthInBars(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
```

- **CHART_WIDTH_IN_PIXELS** 返回图表像素宽度。

```
//+------------------------------------------------------------------+
//| 函数接收图表像素宽度。                                           |
//+------------------------------------------------------------------+
int ChartWidthInPixels(const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   return((int)result);
  }
```

- **CHART_HEIGHT_IN_PIXELS** - 图表像素高度的属性。

```
//+------------------------------------------------------------------+
//| 该函数接收图表的像素高度值。                                     |
//+------------------------------------------------------------------+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
  {
//--- 准备变量获得属性值
   long result=-1;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((int)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置了图表的像素高度值。                                   |
//+------------------------------------------------------------------+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const int sub_windo
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_BACKGROUND** - 图表背景色。

```
//+------------------------------------------------------------------+
//| 该函数接收图表背景色。                                           |
//+------------------------------------------------------------------+
color ChartBackColorGet(const long chart_ID=0)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收图表背景色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置图表背景色。                                           |
//+------------------------------------------------------------------+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置图表背景色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_FOREGROUND** - 坐标轴，比例尺和OHLC线的颜色。

```
//+------------------------------------------------------------------+
//| 该函数接收坐标轴，比例尺和OHLC线的颜色。                         |
//+------------------------------------------------------------------+
color ChartForeColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收坐标轴，比例尺和OHLC线的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置坐标轴，比例尺和OHLC线的颜色。                          |
//+------------------------------------------------------------------+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置坐标轴，比例尺和OHLC线的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_GRID** - 图表网格颜色。

```
//+------------------------------------------------------------------+
//| 这个函数接收图表网格颜色。                                       |
//+------------------------------------------------------------------+
color ChartGridColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//---接收图表网格颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 函数设置图表网格颜色。                                           |
//+------------------------------------------------------------------+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 设置图表网格颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_VOLUME** - 交易量和持仓水平的颜色。

```
//+------------------------------------------------------------------+
//| 该函数接收交易量和市场入场水平                                   |
//| 的颜色。                                                         |
//+------------------------------------------------------------------+
color ChartVolumeColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收交易量和市场入场水平的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_VOLUME,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置交易量和市场入场水平                                   |
//| 的颜色。                                                         |
//+------------------------------------------------------------------+
bool ChartVolumeColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置交易量和市场入场水平的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_VOLUME,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_CHART_UP** -上柱，柱影和牛市蜡烛图主体边缘的颜色。

```
//+------------------------------------------------------------------+
//| 这个函数接收下柱，柱影和熊市蜡烛图主体边缘                       |
//| 的颜色。                                                         |
//+------------------------------------------------------------------+
color ChartUpColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收上柱，柱影和牛市蜡烛图主体边缘的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 这个函数设置上柱，柱影和牛市蜡烛图主体边缘                       |
//| 的颜色。                                                         |
//+------------------------------------------------------------------+
bool ChartUpColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置上柱，柱影和牛市蜡烛图主体边缘的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_CHART_DOWN** - 下柱，柱影和熊市蜡烛图主体边缘的颜色。

```
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 这个函数接收下柱，柱影和熊市蜡烛图主体边缘                                  |
//| 的颜色。                                                                 |
//+------------------------------------------------------------------------+
color ChartDownColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收下柱，柱影和熊市蜡烛图主体边缘的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------------+
//| 这个函数设置下柱，柱影和熊市蜡烛图主体边缘                                  |
//| 的颜色。                                                                 |
//+------------------------------------------------------------------------+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置下柱，柱影和熊市蜡烛图主体边缘的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_CHART_LINE** - 图表线和Doji蜡烛图的颜色。

```
//+------------------------------------------------------------------------+
//| 该函数接收图表线和Doji蜡烛图的颜色。                                      |
//+------------------------------------------------------------------------+
color ChartLineColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 接收图表线和Doji蜡烛图的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置图表线和Doji蜡烛图的颜色。                                |
//+------------------------------------------------------------------+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置图表线和Doji蜡烛图的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BULL** - 牛市蜡烛图主体的颜色。

```
//+------------------------------------------------------------------+
//| 该函数接收牛市蜡烛图主体的颜色。                                    |
//+------------------------------------------------------------------+
color ChartBullColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收牛市蜡烛图主体的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 该函数设置牛市蜡烛图主体的颜色。                                          |
//+------------------------------------------------------------------+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置牛市蜡烛图主体的颜色。
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BEAR** - 熊市蜡烛图主体的颜色。

```
//+------------------------------------------------------------------+
//| 该函数接收熊市蜡烛图主体的颜色。                                          |
//+------------------------------------------------------------------+
color ChartBearColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收熊市蜡烛图主体的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置熊市蜡烛图主体的颜色。                                          |
//+------------------------------------------------------------------+
bool ChartBearColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置熊市蜡烛图主体的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
     {
      //--- 在EA日志显示错误信息
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_BID** - 卖价线的颜色。

```
//+------------------------------------------------------------------+
//| 函数接收卖价线的颜色。                                           |
//+------------------------------------------------------------------+
color ChartBidColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收卖价线的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置卖价线的颜色。                                         |
//+------------------------------------------------------------------+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置卖价线的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_ASK** - 买价线的颜色。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 该函数接收买价线的颜色。                                         |
//+------------------------------------------------------------------+
color ChartAskColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收买价线的颜色
   if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 该函数设置买价线的颜色。                                         |
//+------------------------------------------------------------------+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置买价线的颜色
   if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_LAST** - 最近交易的价格线的颜色（收盘价）。

```
//+------------------------------------------------------------------+
//| 这个函数接收最近交易的价格线的颜色。                             |
//+------------------------------------------------------------------+
color ChartLastColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收最近交易的价格线的颜色（收盘价）
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
//| 这个函数设置最近交易的价格线                                     |
//| 的颜色。                                                         |
//+------------------------------------------------------------------+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置最近交易的价格线的颜色（收盘价）
   if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_COLOR_STOP_LEVEL** - 止损订单水平颜色（止损和获利）。

```
//+------------------------------------------------------------------+
//| 这个函数接收止损和获利水平的颜色。                               |
//+------------------------------------------------------------------+
color ChartStopLevelColorGet(const long chart_ID=0)
  {
//--- 准备变量接收颜色
   long result=clrNONE;
//--- 重置错误的值
   ResetLastError();
//--- 接收止损订单水平的颜色（止损和获利）
   if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return((color)result);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 该函数设置止损和获利水平的颜色。                                          |
//+------------------------------------------------------------------+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置止损订单水平的颜色（止损和获利）
   if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_TRADE_LEVELS** - 在图表上显示交易水平的属性（持仓水平，止损，获利和挂单水平）。

```
//+------------------------------------------------------------------+
//| 这个函数定义是否在图表上显示交易水平。                                    |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用交易水平显示模式。                                        |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_DRAG_TRADE_LEVELS** - 启用使用鼠标在图表上拖拽交易水平的能力的属性

```
//+------------------------------------------------------------------------+
//| 这个函数定义是否允许使用鼠标在图表上拖拽交易                           |
//| 水平。                                                                 |
//+------------------------------------------------------------------------+
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用使用鼠标在图表上拖拽交易水平                      |
//| 的模式。                                                         |
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_DATE_SCALE** - 在图表上显示时间比例的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义是否在图表上显示时间比例。                                |
//+------------------------------------------------------------------+
bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 该函数启用/禁用在图表上显示时间比例的模式。                         |
//+------------------------------------------------------------------+
bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

- **CHART_SHOW_PRICE_SCALE** - 在图表上显示价格比例的属性。

```
//+------------------------------------------------------------------+
//| 该函数定义是否在图表上显示价格比例。                                |
//+------------------------------------------------------------------+
bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
  {
//--- 准备变量获得属性值
   long value;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------+
//| 该函数启用/禁用在图表上显示价格比例的模式。                               |
//+------------------------------------------------------------------------+
bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHOW_ONE_CLICK** - 在图表上显示"一键交易"的属性。

```
//+------------------------------------------------------------------+
//| 检查"单击交易" 面板是否显示在图表上                                 |
//+------------------------------------------------------------------+
bool ChartShowOneClickPanelGet(bool &result,const long chart_ID=0)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 准备变量以获得属性值
   long value;
//--- 重置错误值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在内存中存储图表属性的值
   result=value;
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 启用/禁用显示"单击交易" 面板                                     |
//| 在图表上                                                         |
//+------------------------------------------------------------------+
bool ChartShowOneClickPanelSet(const bool value,const long chart_ID=0)
  {
//--- 重置错误值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_SHIFT_SIZE** - 从右侧边界开始的零柱的百分比值转换大小。

```
//+------------------------------------------------------------------+
//| 该函数接收从图表右侧边界开始的零柱的                             |
//| 百分比值 转换大小 从10%直到50%)。                                |
//+------------------------------------------------------------------+
double ChartShiftSizeGet(const long chart_ID=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
//+------------------------------------------------------------------------------------+
//| 这个函数设置从图表右侧边界开始的零柱的                                             |
//| 百分比值转换大小 从10%直到50%)。要启用该转换                                       |
//| 模式,CHART_SHIFT 属性值应该设置为                                                  |
//| true。                                                                             |
//+------------------------------------------------------------------------------------+
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_FIXED_POSITION** - 图表从左边界以百分比值固定位置。

```
//+------------------------------------------------------------------------+
//| 这个函数接收从左边界以百分比值                                         |
//| 固定位置的图表的位置。                                                 |
//+------------------------------------------------------------------------+
double ChartFixedPositionGet(const long chart_ID=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
```

© 2000-2025, MetaQuotes Ltd.

```
   return(result);
  }
//+------------------------------------------------------------------+
//| 这个函数设置从左边界以百分比值                                   |
//| 固定位置的图表的位置。若要查看固定位置                           |
//| 图表的位置， CHART_AUTOSCROLL                                    |
//| 属性值应该设置为false。                                          |
//+------------------------------------------------------------------+
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_FIXED_MAX** - 表格的固定最大值属性。

```
//+------------------------------------------------------------------+
//| 这个函数接收图表的固定最大值。                                   |
//+------------------------------------------------------------------+
double ChartFixedMaxGet(const long chart_ID=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
//+------------------------------------------------------------------+
//| 这个函数设置图表的固定最大值。                                   |
//| 要改变属性的值，                                                 |
//| CHART_SCALEFIX 属性值应该提前设置为                              |
//| true。                                                           |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_FIXED_MIN** - 图表固定最小值的属性。

```
//+------------------------------------------------------------------+
//| 这个函数接收图表的固定最小值。                                   |
//+------------------------------------------------------------------+
double ChartFixedMinGet(const long chart_ID=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
//+------------------------------------------------------------------+
//| 这个函数设置图表的固定最小值。                                   |
//| 要改变属性的值，                                                 |
//| CHART_SCALEFIX 属性值应该提前设置为                              |
//| true。                                                           |
//+------------------------------------------------------------------+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 在EA日志显示错误信息
       Print(__FUNCTION__+", Error Code = ",GetLastError());
       return(false);
      }
//--- 成功执行
   return(true);
  }
```

- **CHART_POINTS_PER_BAR** - 每柱比例尺的点数值。

```
//+------------------------------------------------------------------------------+
//| 这个函数接收每柱图表比例尺的点数值。                                           |
//+------------------------------------------------------------------------------+
double ChartPointsPerBarGet(const long chart_ID=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
//+------------------------------------------------------------------------------+
//| 这个函数设置每柱的图表比例尺的点数值。                                         |
//| 要查看这个属性值的变化结果，                                                   |
//| 这个                                                                         |
//| CHART_SCALE_PT_PER_BAR 属性值应该提前设置为true。                              |
//+------------------------------------------------------------------------------+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
```

- **CHART_PRICE_MIN** 返回图表的最小值。

```
//+------------------------------------------------------------------------------+
//| 这个函数接收主窗口或子窗口的图表最小值。                                     |
//+------------------------------------------------------------------------------+
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
```

- **CHART_PRICE_MAX** 返回图表的最大值。

```
//+------------------------------------------------------------------------------+
//| 这个函数接收主窗口或子窗口的图表最大值。                                     |
//+------------------------------------------------------------------------------+
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
  {
//--- 准备变量获得结果
   double result=EMPTY_VALUE;
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- 返回图表属性值
   return(result);
  }
```

- **CHART_COMMENT** - 图表评论。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 这个函数接收图表左上角的评论。                                   |
//+------------------------------------------------------------------+
bool ChartCommentGet(string &result,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetString(chart_ID,CHART_COMMENT,result))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数设置图表左上角的评论。                                   |
//+------------------------------------------------------------------+
bool ChartCommentSet(const string str,const long chart_ID=0)
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置属性值
   if(!ChartSetString(chart_ID,CHART_COMMENT,str))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

- **CHART_IS_MAXIMIZED** - 图表窗口最大化。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 定义如果当前图表窗口最大化                                       |
//+------------------------------------------------------------------+
bool ChartWindowsIsMaximized(bool &result,const long chart_ID=0)
  {
//--- 准备接收属性值的变量
   long value;
//--- 重置错误值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_IS_MAXIMIZED))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在变量中存储图表属性值
   result=value;
//--- 成功执行
   return(true);
  }
```

- **CHART_IS_MINIMIZED** - 图表窗口最小化。

```
//+------------------------------------------------------------------+
//| 定义如果当前图表窗口最小化                                       |
//+------------------------------------------------------------------+
bool ChartWindowsIsMinimized(bool &result,const long chart_ID=0)
  {
//--- 准备接收属性值的变量
   long value;
//--- 重置错误值
   ResetLastError();
//--- 接收属性值
   if(!ChartGetInteger(chart_ID,CHART_IS_MINIMIZED))
     {
      //--- 在EA日志显示错误信息
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- 在变量中存储图表属性值
   result=value;
//--- 成功执行
   return(true);
  }
```

图表属性面板

```
//--- 连接控制元素的程序库
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- 预定义常量
#define X_PROPERTY_NAME_1     10  //第一列属性名的x坐标
#define X_PROPERTY_VALUE_1    225 //第一列属性值的 x 坐标
```

© 2000-2025, MetaQuotes Ltd.

```
#define X_PROPERTY_NAME_2     345 // 第二和第三列的属性名的x坐标
#define X_PROPERTY_VALUE_2    550 // 第二和第三列的属性值的x坐标
#define X_BUTTON_1            285 // 第一列按钮的x坐标
#define X_BUTTON_2            700 //第二列按钮的 x 坐标
#define Y_PROPERTY_1          30  // 第一和第二列开始的y 坐标
#define Y_PROPERTY_2          286 // 第三列开始的y坐标
#define Y_DISTANCE            16  // y 轴的行间距
#define LAST_PROPERTY_NUMBER 111 // 最近图形的属性号
//--- 输入参数
input color InpFirstColor=clrDodgerBlue; // 奇数行的颜色
input color InpSecondColor=clrGoldenrod; // 偶数行的颜色
//--- 变量和数组
CChartObjectLabel  ExtLabelsName[];  // 显示属性名的标签
CChartObjectLabel  ExtLabelsValue[]; // 显示属性值的标签
CChartObjectButton ExtButtons[];     // 按钮
int                ExtNumbers[];     // 属性指数
string             ExtNames[];       // 属性名称
uchar              ExtDataTypes[];   // 属性数据类型 整型，双精度，字符串）
uint               ExtGroupTypes[];  // 存储有关属性归属于一个组的数据的数组
uchar              ExtDrawTypes[];   // 存储有关属性显示类型的数据的数组
double             ExtMaxValue[];    // 面板工作时最大可能的属性值
double             ExtMinValue[];    // 面板工作时最小可能的属性值
double             ExtStep[];        // 更改属性的步骤
int                ExtCount;         // 所有属性的总数
color              ExtColors[2];     // 用于显示行的颜色的数组
string             ExtComments[2];   // 评论数组 用于CHART_COMMENT 属性）
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 在图表上显示评论
   Comment("SomeComment");
//--- 数组中存储颜色以便过后能够在它们之间切换
   ExtColors[0]=InpFirstColor;
   ExtColors[1]=InpSecondColor;
//--- 数组中存储评论以便过后在它们之间切换
   ExtComments[0]="FirstComment";
   ExtComments[1]="SecondComment";
//--- 准备并显示管理图表属性的控制面板
   if(!PrepareControls())
      return(INIT_FAILED);
//--- 成功执行
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA的去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 在图表上移除评论
   Comment("");
  }
//+------------------------------------------------------------------+
//| 图表事件处理程序                                                 |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 检查点击图表对象的事件
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      //--- 用分隔符分开对象名称
      string obj_name[];
      StringSplit(sparam,'_',obj_name);
      //--- 检查对象是否是个按钮
      if(obj_name[0]=="Button")
        {
         //--- 接收按钮指数
         int index=(int)StringToInteger(obj_name[1]);
         //--- 不按按钮
         ExtButtons[index].State(false);
         //--- 根据类型设置新的属性值
         if(ExtDataTypes[index]=='I')
            ChangeIntegerProperty(index);
         if(ExtDataTypes[index]=='D')
            ChangeDoubleProperty(index);
         if(ExtDataTypes[index]=='S')
            ChangeStringProperty(index);
        }
     }
//--- 重画属性值
   RedrawProperties();
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 改变图表的整型属性                                               |
//+------------------------------------------------------------------+
void ChangeIntegerProperty(const int index)
  {
//--- 接收当前属性值
   long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index]);
//--- 定义以下属性值
   switch(ExtDrawTypes[index])
     {
      case 'C':
```

© 2000-2025, MetaQuotes Ltd.

```
         value=GetNextColor((color)value);
         break;
      default:
         value=(long)GetNextValue((double)value,index);
         break;
     }
//--- 设置新的属性值
   ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,value);
  }
//+------------------------------------------------------------------+
//| 改变图表的双精度属性                                             |
//+------------------------------------------------------------------+
void ChangeDoubleProperty(const int index)
  {
//--- 接收当前属性值
   double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index]);
//--- 定义以下属性值
   value=GetNextValue(value,index);
//--- 设置新的属性值
   ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
  }
//+------------------------------------------------------------------+
//| 改变图表的字符串属性                                             |
//+------------------------------------------------------------------+
void ChangeStringProperty(const int index)
  {
//--- 用于 ExtComments数组内转换的静态变量
   static uint comment_index=1;
//--- 改变接收另一个评论的指数
   comment_index=1-comment_index;
//--- 设置新的属性值
   ChartSetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComments[comment_
  }
//+------------------------------------------------------------------+
//| 定义下一个属性值                                                 |
//+------------------------------------------------------------------+
double GetNextValue(const double value,const int index)
  {
   if(value+ExtStep[index]<=ExtMaxValue[index])
      return(value+ExtStep[index]);
   else
      return(ExtMinValue[index]);
  }
//+------------------------------------------------------------------+
//| 接收颜色类型属性的下一个颜色                                     |
//+------------------------------------------------------------------+
color GetNextColor(const color clr)
  {
//--- 返回以下颜色值
```

© 2000-2025, MetaQuotes Ltd.

```
   switch(clr)
     {
      case clrWhite: return(clrRed);
      case clrRed:   return(clrGreen);
      case clrGreen: return(clrBlue);
      case clrBlue:  return(clrBlack);
      default:       return(clrWhite);
     }
  }
//+------------------------------------------------------------------+
//| 重画属性值                                                       |
//+------------------------------------------------------------------+
void RedrawProperties(void)
  {
//--- 属性值文本
   string text;
   long   value;
//--- 循环的属性数量
   for(int i=0;i<ExtCount;i++)
     {
      text="";
      switch(ExtDataTypes[i])
        {
         case 'I':
            //--- 接收当前属性值
            if(!ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[i],0,value))
            break;
            //--- 整型属性文本
            switch(ExtDrawTypes[i])
              {
               //--- 颜色属性
               case 'C':
                  text=(string)((color)value);
                  break;
                  //--- 布尔属性
               case 'B':
                  text=(string)((bool)value);
                  break;
                  //--- ENUM_CHART_MODE 枚举属性
               case 'M':
                  text=EnumToString((ENUM_CHART_MODE)value);
                  break;
                  //--- ENUM_CHART_VOLUME_MODE 枚举属性
               case 'V':
                  text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                  break;
                  //--- int 类型号
               default:
                  text=IntegerToString(value);
```

© 2000-2025, MetaQuotes Ltd.

```
                  break;
               }
            break;
         case 'D':
            //--- 双精度属性文本
            text=DoubleToString(ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumber
            break;
         case 'S':
            //--- 字符串属性文本
            text=ChartGetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[i]);
            break;
        }
       //--- 显示属性值
       ExtLabelsValue[i].Description(text);
      }
   }
//+------------------------------------------------------------------+
//| 创建管理图表属性面板                                             |
//+------------------------------------------------------------------+
bool PrepareControls(void)
   {
//--- 为数组存储分配内存
   MemoryAllocation(LAST_PROPERTY_NUMBER+1);
//--- 变量
   int i=0;      // 循环变量
   int col_1=0; // 第一列的属性数
   int col_2=0; // 第二列的属性数
   int col_3=0; // 第三列的属性数
//--- 当前属性数 - 0
   ExtCount=0;
//--- 在循环中查找属性
   while(i<=LAST_PROPERTY_NUMBER)
     {
      //--- 存储当前属性数
      ExtNumbers[ExtCount]=i;
      //--- 增加循环变量值
      i++;
      //--- 检查是否有这个数的属性
      if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[ExtCount],Ex
        {
         //--- 创建属性的控制元素
         switch(ExtGroupTypes[ExtCount])
           {
            case 1:
               //--- 创建属性标签和按钮
               if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VALUE_1,X_BUTT
               return(false);
               //--- 第一列的元素数已经增加
               col_1++;
```

© 2000-2025, MetaQuotes Ltd.

```
                break;
            case 2:
                //--- 创建属性标签和按钮
                if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,X_BUTT
                return(false);
                //--- 第二列的元素数已经增加
                col_2++;
                break;
            case 3:
                //--- 只创建属性标签
                if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,0,Y_PR
                return(false);
                //--- 第三列的元素数已经增加
                col_3++;
                break;
           }
         //--- 定义最大最小属性值和步骤
         GetMaxMinStep(ExtNumbers[ExtCount],ExtMaxValue[ExtCount],ExtMinValue[ExtCount
         //--- 增加属性数量
         ExtCount++;
        }
     }
//--- 释放数组不用的内存
   MemoryAllocation(ExtCount);
//--- 重画属性值
   RedrawProperties();
   ChartRedraw();
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 为数组分配内存                                                   |
//+------------------------------------------------------------------+
void MemoryAllocation(const int size)
  {
   ArrayResize(ExtLabelsName,size);
   ArrayResize(ExtLabelsValue,size);
   ArrayResize(ExtButtons,size);
   ArrayResize(ExtNumbers,size);
   ArrayResize(ExtNames,size);
   ArrayResize(ExtDataTypes,size);
   ArrayResize(ExtGroupTypes,size);
   ArrayResize(ExtDrawTypes,size);
   ArrayResize(ExtMaxValue,size);
   ArrayResize(ExtMinValue,size);
   ArrayResize(ExtStep,size);
  }
//+------------------------------------------------------------------+
//| 检查是否属性指数属于                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//| ENUM_CHART_PROPERTIES 枚举                                       |
//+------------------------------------------------------------------+
bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_type,uchar &d
  {
//--- 检查属性是否是整型类型的属性
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
   if(_LastError==0)
     {
      data_type='I';                          // 来自 ENUM_CHART_PROPERTY_INTEGER 枚举的属性
      GetTypes(ind,group_type,draw_type); // 定义属性显示参数
      return(true);
     }
//--- 检查属性是否是双精度类型
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
   if(_LastError==0)
     {
      data_type='D';                          // 来自 ENUM_CHART_PROPERTY_DOUBLE 枚举的属性
      GetTypes(ind,group_type,draw_type); // 定义属性显示参数
      return(true);
     }
//--- 检查属性是否是字符串类型
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
   if(_LastError==0)
     {
      data_type='S';                          // 来自 ENUM_CHART_PROPERTY_STRING 枚举的属性
      GetTypes(ind,group_type,draw_type); // 定义属性显示参数
      return(true);
     }
//--- 属性不属于任何枚举
   return(false);
  }
//+------------------------------------------------------------------+
//| 定义属性应该存储的组，                                           |
//| 以及其展示类型                                                   |
//+------------------------------------------------------------------+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- 检查属性是否属于第三组
//--- 第三组属性显示在从CHART_BRING_TO_TOP开始的第二列
   if(CheckThirdGroup(property_number,group_type,draw_type))
      return;
//--- 检查属性是否属于第二组
//--- 第二组属性显示在第二列的开头
   if(CheckSecondGroup(property_number,group_type,draw_type))
      return;
//--- 如果您在这里找到了自己，属性属于第一组 (第一列)
```

© 2000-2025, MetaQuotes Ltd.

```
   CheckFirstGroup(property_number,group_type,draw_type);
  }
//+------------------------------------------------------------------+
//| 这个函数检查属性是否属于第三组并                                 |
//| 在肯定回答时定义显示类型                                         |
//+------------------------------------------------------------------+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- 检查属性是否属于第三组
   switch(property_number)
     {
      //--- 布尔属性
      case CHART_IS_OBJECT:
      case CHART_WINDOW_IS_VISIBLE:
         draw_type='B';
         break;
         //--- 整型属性
      case CHART_VISIBLE_BARS:
      case CHART_WINDOWS_TOTAL:
      case CHART_WINDOW_HANDLE:
      case CHART_WINDOW_YDISTANCE:
      case CHART_FIRST_VISIBLE_BAR:
      case CHART_WIDTH_IN_BARS:
      case CHART_WIDTH_IN_PIXELS:
         draw_type='I';
         break;
         //--- 双精度属性
      case CHART_PRICE_MIN:
      case CHART_PRICE_MAX:
         draw_type='D';
         break;
         //--- 实际上，这个属性实在所有其他的上面显示图表的命令
         //--- 无需应用这个面板，因为窗口会始终保持
         //--- 在其他的上面，在我们使用之前
      case CHART_BRING_TO_TOP:
         draw_type=' ';
         break;
         //--- 属性不属于第三组
      default:
         return(false);
     }
//--- 属性属于第三组
   group_type=3;
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数检查属性是否属于第二组并                                 |
//| 在肯定回答时定义显示类型                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- 检查属性是否属于第二组
   switch(property_number)
     {
      //--- ENUM_CHART_MODE 类型属性
      case CHART_MODE:
         draw_type='M';
         break;
         //--- ENUM_CHART_VOLUME_MODE 类型属性
      case CHART_SHOW_VOLUMES:
         draw_type='V';
         break;
         //--- 字符串属性
      case CHART_COMMENT:
         draw_type='S';
         break;
         //--- 颜色属性
      case CHART_COLOR_BACKGROUND:
      case CHART_COLOR_FOREGROUND:
      case CHART_COLOR_GRID:
      case CHART_COLOR_VOLUME:
      case CHART_COLOR_CHART_UP:
      case CHART_COLOR_CHART_DOWN:
      case CHART_COLOR_CHART_LINE:
      case CHART_COLOR_CANDLE_BULL:
      case CHART_COLOR_CANDLE_BEAR:
      case CHART_COLOR_BID:
      case CHART_COLOR_ASK:
      case CHART_COLOR_LAST:
      case CHART_COLOR_STOP_LEVEL:
         draw_type='C';
         break;
         //--- 属性不属于第二组
      default:
         return(false);
     }
//--- 属性属于第二组
   group_type=2;
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数被调用只有在已知                                         |
//| 属性不属于第二和第三属性组时才可以                               |
//+------------------------------------------------------------------+
void CheckFirstGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- 属性属于第一组
   group_type=1;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义属性显示类型
   switch(property_number)
     {
      //--- 整型属性
      case CHART_SCALE:
      case CHART_HEIGHT_IN_PIXELS:
         draw_type='I';
         return;
         //--- 双精度属性
      case CHART_SHIFT_SIZE:
      case CHART_FIXED_POSITION:
      case CHART_FIXED_MAX:
      case CHART_FIXED_MIN:
      case CHART_POINTS_PER_BAR:
         draw_type='D';
         return;
         //--- 只保留布尔属性
      default:
         draw_type='B';
         return;
     }
  }
//+------------------------------------------------------------------+
//| 创建属性标签和按钮                                               |
//+------------------------------------------------------------------+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
  {
//--- 用于ExtColors颜色数组内转换的静态数组
   static uint color_index[3]={1,1,1};
//--- 改变指数，接收另一个颜色
   color_index[type]=1-color_index[type];
//--- 显示属性的标签和按钮（如果 btn=true）
   if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],ExtColors[colo
      return(false);
   if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[color_index[t
      return(false);
   if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
      return(false);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建标签                                                         |
//+------------------------------------------------------------------+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string text,
                 const color clr,const int x,const int y)
  {
   if(!lbl.Create(0,"Label_"+name,0,x,y)) return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!lbl.Description(text))              return(false);
   if(!lbl.FontSize(10))                   return(false);
   if(!lbl.Color(clr))                     return(false);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建按钮                                                         |
//+------------------------------------------------------------------+
bool ButtonCreate(CChartObjectButton &btn,const string name,
                  const int x,const int y)
  {
   if(!btn.Create(0,"Button_"+name,0,x,y,50,15)) return(false);
   if(!btn.Description("Next"))                  return(false);
   if(!btn.FontSize(10))                         return(false);
   if(!btn.Color(clrBlack))                      return(false);
   if(!btn.BackColor(clrWhite))                  return(false);
   if(!btn.BorderColor(clrBlack))                return(false);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 定义最大和最小属性值和步骤                                       |
//+------------------------------------------------------------------+
void GetMaxMinStep(const int property_number,double &max,double &min,double &step)
  {
   double value;
//--- 根据属性类型设置值
   switch(property_number)
     {
      case CHART_SCALE:
         max=5;
         min=0;
         step=1;
         break;
      case CHART_MODE:
      case CHART_SHOW_VOLUMES:
         max=2;
         min=0;
         step=1;
         break;
      case CHART_SHIFT_SIZE:
         max=50;
         min=10;
         step=2.5;
         break;
      case CHART_FIXED_POSITION:
         max=90;
         min=0;
```

© 2000-2025, MetaQuotes Ltd.

```
         step=15;
         break;
      case CHART_POINTS_PER_BAR:
         max=19;
         min=1;
         step=3;
         break;
      case CHART_FIXED_MAX:
         value=ChartGetDouble(0,CHART_FIXED_MAX);
         max=value*1.25;
         min=value;
         step=value/32;
         break;
      case CHART_FIXED_MIN:
         value=ChartGetDouble(0,CHART_FIXED_MIN);
         max=value;
         min=value*0.75;
         step=value/32;
         break;
      case CHART_HEIGHT_IN_PIXELS:
         max=700;
         min=520;
         step=30;
         break;
         //--- 默认值
      default:
         max=1;
         min=0;
         step=1;
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# 对象常量

可以创建44个图解对象并在价格图表中显示出来，所有与对象有关的常量都被分成9组：

- 对象类型 -图解对象的标识符；
- 对象属性 -建立和获得图解对象的性质；
- 对象定位方法 -在图表中定位对象常量；
- 定位角 -指出时间表的角度，在对象上定位；
- 对象可见性 -对象可见时建立时间表；
- 埃利奥特波动水平 -波浪渐变标记；
- 江恩对象 -江恩角度线盒网格的趋向常量；
- 网页颜色 -预定义网络颜色的常量；
- Wingdings -W ingdings字形的字符代码

© 2000-2025, MetaQuotes Ltd.

# 物件类型

当使用ObjectCreate()函数建立图表对象，对于增加的对象类型有必要说明一下，它有可能是ENUM_OBJECT计算式中的一个，对象的下一项规格属性有可能使用与图解对象一同使用

**ENUM_OBJECT**

| ID | | 描述 |
|---|---|---|
| OBJ_VLINE |  | 垂直线 |
| OBJ_HLINE |  | 水平线 |
| OBJ_TREND | | 趋势线 |
| OBJ_TRENDBYANGLE | | 趋势角度 |
| OBJ_CYCLES | | 周期线 |
| OBJ_ARROWED_LINE | | 箭头线路 |
| OBJ_CHANNEL | | 等距离通道 |
| OBJ_STDDEVCHANNEL | | 标准偏离通道 |
| OBJ_REGRESSION | | 线形回归通道 |
| OBJ_PITCHFORK | | 安德鲁分叉线 |
| OBJ_GANNLINE | | 江恩线 |
| OBJ_GANNFAN | | 江恩扇形线 |
| OBJ_GANNGRID | | 江恩网格线 |
| OBJ_FIBO | | 斐波纳契撤回 |
| OBJ_FIBOTIMES | | 斐波纳契时间周期线 |
| OBJ_FIBOFAN | | 斐波纳契扇形线 |
| OBJ_FIBOARC | | 斐波纳契弧线 |
| OBJ_FIBOCHANNEL | | 斐波纳契通道 |
| OBJ_EXPANSION | | 斐波纳契扩展 |
| OBJ_ELLIOTWAVE5 | | 埃利奥特动机波动 |
| OBJ_ELLIOTWAVE3 | | 埃利奥特修正波动 |
| OBJ_RECTANGLE | | 巨型 |
| OBJ_TRIANGLE | | 三角形 |
| OBJ_ELLIPSE | | 椭圆形 |
| OBJ_ARROW_THUMB_UP | | 大拇指向上 |
| OBJ_ARROW_THUMB_DOWN | | 大拇指向下 |

© 2000-2025, MetaQuotes Ltd.

| ID | | 描述 |
|---|---|---|
| OBJ_ARROW_UP | | 向上箭头 |
| OBJ_ARROW_DOWN | | 向下箭头 |
| OBJ_ARROW_STOP | | 停止标志 |
| OBJ_ARROW_CHECK | | 检测标志 |
| OBJ_ARROW_LEFT_PRICE | | 左侧定价标签 |
| OBJ_ARROW_RIGHT_PRICE | | 右侧定价标签 |
| OBJ_ARROW_BUY | | 买入标志 |
| OBJ_ARROW_SELL | | 卖出标志 |
| OBJ_ARROW | | 箭头 |
| OBJ_TEXT | | 文本 |
| OBJ_LABEL | | 标签 |
| OBJ_BUTTON | | 按钮 |
| OBJ_CHART | | 图表 |
| OBJ_BITMAP | | 位图 |
| OBJ_BITMAP_LABEL | | 位图标签 |
| OBJ_EDIT | | 编辑 |
| OBJ_EVENT | | “事件”对象在经济日历对应一个事件 |
| OBJ_RECTANGLE_LABEL | | 用于创建和设计自定义图形界面的“矩形标签”对象。 |

© 2000-2025, MetaQuotes Ltd.

## OBJ_VLINE

垂直线。

注意

绘制垂直线时，可以设置所有图表窗口的线显示模式（属性 OBJPROP_RAY）。

示例

下面的脚本创建和移动图表上的垂直线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Vertical Line\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="VLine";      // 线的名称
input int              InpDate=25;           // 事件日期，%
input color            InpColor=clrRed;      // 线的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // 线的风格
input int              InpWidth=3;           // 线的宽度
input bool             InpBack=false;        // 背景线
input bool             InpSelection=true;    // 突出移动
input bool             InpRay=true;          // 线延续下降
input bool             InpHidden=true;       // 隐藏在对象列表
input long             InpZOrder=0;          // 鼠标单击优先
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 创建垂直线                                                       |
//+------------------------------------------------------------------+
bool VLineCreate(const long            chart_ID=0,        // 图表 ID
                 const string          name="VLine",      // 线的名称
                 const int             sub_window=0,      // 子窗口指数
                 datetime              time=0,            // 线的时间
                 const color           clr=clrRed,        // 线的颜色
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // 线的风格
                 const int             width=1,           // 线的宽度
                 const bool            back=false,        // 在背景中
                 const bool            selection=true,    // 突出移动
                 const bool            ray=true,          // 线延续下降
                 const bool            hidden=true,       // 隐藏在对象列表
                 const long            z_order=0)         // 鼠标单击优先
  {
//--- 如果没有线的时间，通过收盘柱来绘制它
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 创建垂直线
   if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create a vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) the mode of displaying the line in the chart subwindow
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动垂直线                                                       |
//+------------------------------------------------------------------+
bool VLineMove(const long   chart_ID=0,   // 图表 ID
               const string name="VLine", // 线的名称
               datetime     time=0)       // 线的时间
  {
//--- 如果没有设置线的时间，将线移动到收盘柱
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 移动垂直线
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除垂直线                                                       |
//+------------------------------------------------------------------+
bool VLineDelete(const long   chart_ID=0,   // 图表 ID
                 const string name="VLine") // 线的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除垂直线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Error! Incorrect values of input parameters!");
       return;
      }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 存储要使用的日期值的数组
//--- 设置和改变线定位点的坐标
   datetime date[];
//--- 内存分配
   ArrayResize(date,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 定义画线的点
   int d=InpDate*(bars-1)/100;
//--- 创建垂直线
   if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpRay,InpHidden,InpZOrder))
      return;
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动线
//--- 循环计数器
   int h_steps=bars/2;
//--- 移动线
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!VLineMove(0,InpName,date[d]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.03 秒延迟
      Sleep(30);
     }
//--- 1 秒延迟
   Sleep(1000);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 删除图表通道
   VLineDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

## OBJ_HLINE

水平线。

示例

下面的脚本创建和移动图表上的水平线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Horizontal Line\" graphical object."
#property description "Anchor point price is set in percentage of the height of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="HLine";      // 线的名称
input int              InpPrice=25;          // 线型价格, %
input color            InpColor=clrRed;      // 线的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // 线的风格
input int              InpWidth=3;           // 线的宽度
input bool             InpBack=false;        // 背景线
input bool             InpSelection=true;    // 突出移动
input bool             InpHidden=true;       // 隐藏在对象列表
input long             InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建水平线                                                       |
//+------------------------------------------------------------------+
bool HLineCreate(const long            chart_ID=0,        // 图表 ID
```

© 2000-2025, MetaQuotes Ltd.

```
                  const string          name="HLine",       // 线的名称
                  const int             sub_window=0,       // 子窗口指数
                  double                price=0,            // 线的价格
                  const color           clr=clrRed,         // 线的颜色
                  const ENUM_LINE_STYLE style=STYLE_SOLID,  // 线的风格
                  const int             width=1,            // 线的宽度
                  const bool            back=false,         // 在背景中
                  const bool            selection=true,     // 突出移动
                  const bool            hidden=true,        // 隐藏在对象列表
                  const long            z_order=0)          // 鼠标单击优先
  {
//--- 如果没有设置价格，则在当前卖价水平设置它
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 创建水平线
   if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
     {
      Print(__FUNCTION__,
            ": failed to create a horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动水平线                                                       |
//+------------------------------------------------------------------+
bool HLineMove(const long   chart_ID=0,   // 图表 ID
               const string name="HLine", // 线的名称
```

© 2000-2025, MetaQuotes Ltd.

```
                double       price=0)      // 线的价格
  {
//--- 如果没有设置线的价格，则将其移动到当前卖价水平
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动水平线
   if(!ObjectMove(chart_ID,name,0,0,price))
     {
      Print(__FUNCTION__,
            ": failed to move the horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除水平线                                                       |
//+------------------------------------------------------------------+
bool HLineDelete(const long   chart_ID=0,   // 图表ID
                 const string name="HLine") // 线的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除水平线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的价格值的数组
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置和改变线定位点的坐标
   double price[];
//--- 内存分配
   ArrayResize(price,accuracy);
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义画线的点
   int p=InpPrice*(accuracy-1)/100;
//--- 创建水平线
   if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动线
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动线
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p<accuracy-1)
         p+=1;
      //--- 移动点
      if(!HLineMove(0,InpName,price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除
   HLineDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_TREND

趋势线。

注意

对于趋势线，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 根据属性）。

示例

下面的脚本创建和移动图表上的趋势线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Trend Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Trend";      // 线的名称
input int              InpDate1=35;          // 第1个点的日期，%
input int              InpPrice1=60;         // 第1个点的价格，%
input int              InpDate2=65;          // 第2个点的日期，%
input int              InpPrice2=40;         // 第2个点的价格，%
input color            InpColor=clrRed;      // 线的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // 线的风格
input int              InpWidth=2;           // 线的宽度
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpBack=false;        // 背景线
input bool              InpSelection=true;    // 突出移动
input bool              InpRayLeft=false;     // 线延续向左
input bool              InpRayRight=false;    // 线延续向右
input bool              InpHidden=true;       // 隐藏在对象列表
input long              InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建趋势线                                         |
//+------------------------------------------------------------------+
bool TrendCreate(const long            chart_ID=0,        // 图表 ID
                 const string          name="TrendLine",  // 线的名称
                 const int             sub_window=0,      // 子窗口指数
                 datetime              time1=0,           // 第一个点的时间
                 double                price1=0,          // 第一个点的价格
                 datetime              time2=0,           // 第二个点的时间
                 double                price2=0,          // 第二个点的价格
                 const color           clr=clrRed,        // 线的颜色
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // 线的风格
                 const int             width=1,           // 线的宽度
                 const bool            back=false,        // 在背景中
                 const bool            selection=true,    // 突出移动
                 const bool            ray_left=false,    // 线延续向左
                 const bool            ray_right=false,   // 线延续向右
                 const bool            hidden=true,       // 隐藏在对象列表
                 const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeTrendEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建趋势线
   if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动趋势线定位点                                                 |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,        // 图表 ID
                      const string name="TrendLine", // 线的名称
                      const int    point_index=0,    // 定位点指数
                      datetime     time=0,           // 定位点时间坐标
                      double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动趋势线定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数删除图表的趋势线                                         |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,        // chart's ID
                 const string name="TrendLine") // 线的颜色
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除趋势线
   if(!ObjectDelete(chart_ID,name))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print(__FUNCTION__,
             ": failed to delete a trend line! Error code = ",GetLastError());
       return(false);
      }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查趋势线定位点的值和为空点设置                                 |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二点的价格没有设置，则它与第一点的价格相等
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
```

© 2000-2025, MetaQuotes Ltd.

```
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变线定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义画线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建趋势线
   if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,InpStyle
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动线的定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 垂直移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 垂直移动第二个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p2<accuracy-1)
         p2+=1;
      //--- 移动点
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 半秒延迟
   Sleep(500);
//--- 循环计数器
   int h_steps=bars/2;
//--- 同时水平移动两个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- 切换点
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.03 秒延迟
      Sleep(30);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除趋势线
```

© 2000-2025, MetaQuotes Ltd.

```
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_TRENDBYANGLE

角度趋势线。

注意

对于角度趋势线，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 根据属性）。

角度和第二个定位点的坐标都可以用于设置线的倾斜度。

示例

下面的脚本创建和移动图表上的趋势线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Trend Line By Angle\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Trend";      // 线的名称
input int              InpDate1=50;          // 第1个点的日期，%
input int              InpPrice1=75;         // 第1个点的价格，%
input int              InpAngle=0;           // 线型倾斜角
input color            InpColor=clrRed;      // 线的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // 线的风格
input int              InpWidth=2;           // 线的宽度
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpBack=false;        // 背景线
input bool              InpSelection=true;    // 突出移动
input bool              InpRayLeft=false;     // 线延续向左
input bool              InpRayRight=true;     // 线延续向右
input bool              InpHidden=true;       // 隐藏在对象列表
input long              InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建角度趋势线                                                   |
//+------------------------------------------------------------------+
bool TrendByAngleCreate(const long              chart_ID=0,         // 图表 ID
                        const string            name="TrendLine",   // 线的名称
                        const int               sub_window=0,       // 子窗口指数
                        datetime                time=0,             // 点的时间
                        double                  price=0,            // 点的价格
                        const double            angle=45.0,         // 斜角
                        const color             clr=clrRed,         // 线的颜色
                        const ENUM_LINE_STYLE   style=STYLE_SOLID,  // 线的风格
                        const int               width=1,            // 线的宽度
                        const bool              back=false,         // 在背景中
                        const bool              selection=true,     // 突出移动
                        const bool              ray_left=false,     // 线延续向左
                        const bool              ray_right=true,     // 线延续向右
                        const bool              hidden=true,        // 隐藏在对象列表
                        const long              z_order=0)          // 鼠标单击优先
  {
//--- 创建第二个点以便于鼠标拖拽趋势线
   datetime time2=0;
   double   price2=0;
//--- 若未设置则设置定位点的坐标
   ChangeTrendEmptyPoints(time,price,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 使用2个点创建趋势线
   if(!ObjectCreate(chart_ID,name,OBJ_TRENDBYANGLE,sub_window,time,price,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- 改变趋势线的斜角;改变斜角时，第二点的坐标
//--- 根据新的角度值，自动重新定义线的点
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变趋势线定位点的坐标                                           |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,        // 图表 ID
                      const string name="TrendLine", // 线的名称
                      datetime     time=0,           // 定位点时间坐标
                      double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动趋势线定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变趋势线的斜角                                                 |
//+------------------------------------------------------------------+
bool TrendAngleChange(const long   chart_ID=0,        // 图表 ID
                      const string name="TrendLine", // 趋势线名称
                      const double angle=45)         // 趋势线斜角
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变趋势线的斜角
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change the line's slope angle! Error code = ",GetLastError())
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除趋势线                                                       |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,        // 图表 ID
                 const string name="TrendLine") // 线的颜色
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除趋势线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查趋势线定位点的值和为空点设置                                 |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 设置第二坐标，辅助点
//--- 第二个点是左侧第9个柱并且价格相同
   datetime second_point_time[10];
   CopyTime(Symbol(),Period(),time1,10,second_point_time);
   time2=second_point_time[0];
```

© 2000-2025, MetaQuotes Ltd.

```
   price2=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变线定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义画线的点
   int d1=InpDate1*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- 创建趋势线
   if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor,InpStyle,
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动和旋转线
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动定位点并改变线的斜角
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!TrendPointChange(0,InpName,date[d1],price[p1]))
         return;
      if(!TrendAngleChange(0,InpName,18*(i+1)))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_CYCLES

循环线。

注意

线之间的距离通过对象两个定位点的时间坐标来设置。

示例

下面的脚本创建和移动图表上的循环线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates cycle lines on the chart."
#property description "Anchor point coordinates are set in percentage"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Cycles";    // 对象的名称
input int              InpDate1=10;         // 第1个点的日期，%
input int              InpPrice1=45;        // 第1个点的价格，%
input int              InpDate2=20;         // 第2个点的日期，%
input int              InpPrice2=55;        // 第2个点的价格，%
input color            InpColor=clrRed;     //循环线的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DOT;  // 循环线的风格
input int              InpWidth=1;          // 循环线的宽度
input bool             InpBack=false;       // 背景对象
```

© 2000-2025, MetaQuotes Ltd.

```
input bool             InpSelection=true;  // 突出移动
input bool             InpHidden=true;     //隐藏在对象列表
input long             InpZOrder=0;        // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建循环线                                                       |
//+------------------------------------------------------------------+
bool CyclesCreate(const long             chart_ID=0,        // 图表 ID
                  const string           name="Cycles",     // 对象名称
                  const int              sub_window=0,      // 子窗口指数
                  datetime               time1=0,           // 第一个点的时间
                  double                 price1=0,          // 第一个点的价格
                  datetime               time2=0,           // 第二个点的时间
                  double                 price2=0,          // 第二个点的价格
                  const color            clr=clrRed,        // 循环线的颜色
                  const ENUM_LINE_STYLE  style=STYLE_SOLID, // 循环线的风格
                  const int              width=1,           // 循环线的宽度
                  const bool             back=false,        // 在背景中
                  const bool             selection=true,    // 突出移动
                  const bool             hidden=true,       // 隐藏在对象列表
                  const long             z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建循环线
   if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create cycle lines! Error code = ",GetLastError());
      return(false);
     }
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool CyclesPointChange(const long   chart_ID=0,    // 图表ID
                       const string name="Cycles", // 对象的名称
                       const int    point_index=0, // 定位点指数
                       datetime     time=0,        // 定位点时间坐标
                       double       price=0)       // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除循环线                                                       |
//+------------------------------------------------------------------+
bool CyclesDelete(const long   chart_ID=0,    // 图表 ID
                  const string name="Cycles") // 对象的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除循环线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete cycle lines! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 检查循环线定位点的值和为空点设置                                          |
//| 默认的值                                                                 |
//+------------------------------------------------------------------------+
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二点的价格没有设置，则它与第一点的价格相等
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变循环线定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义画循环线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建趋势线
   if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int h_steps=bars/5;
//--- 移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      //--- 移动点
      if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
```

© 2000-2025, MetaQuotes Ltd.

```
   Sleep(1000);
//--- 循环计数器
   h_steps=bars/4;
//--- 移动第一个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      //--- 移动点
      if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   CyclesDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROWED_LINE

箭头线。

示例

下面的脚本创建和移动图表上的箭头线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Arrowed line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="ArrowedLine"; // 线的名称
input int              InpDate1=35;           // 第1个点的日期，%
input int              InpPrice1=60;          // 第1个点的价格，%
input int              InpDate2=65;           // 第2个点的日期，%
input int              InpPrice2=40;          // 第2个点的价格，%
input color            InpColor=clrRed;       // 线的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;   // 线的风格
input int              InpWidth=2;            // 线的宽度
input bool             InpBack=false;         // 背景线
input bool             InpSelection=true;     // 突出移动
input bool             InpHidden=true;        // 隐藏在对象列表
input long             InpZOrder=0;           // 鼠标单击优先
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 通过已给的坐标创建箭头线                                                  |
//+------------------------------------------------------------------+
bool ArrowedLineCreate(const long            chart_ID=0,          // 图表 ID
                       const string          name="ArrowedLine", // 线的名称
                       const int             sub_window=0,       // 子窗口指数
                       datetime              time1=0,            // 第一个点的时间
                       double                price1=0,           // 第一个点的价格
                       datetime              time2=0,            // 第二个点的时间
                       double                price2=0,           // 第二个点的价格
                       const color           clr=clrRed,         // 线的颜色
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // 线的风格
                       const int             width=1,            // 线的宽度
                       const bool            back=false,         // 在背景中
                       const bool            selection=true,     // 突出移动
                       const bool            hidden=true,        // 隐藏在对象列表
                       const long            z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowedLineEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建箭头线
   if(!ObjectCreate(chart_ID,name,OBJ_ARROWED_LINE,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": failed to create an arrowed line! Error code = ",GetLastError());
      return(false);
     }
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动线的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 移动箭头线的定位点                                                 |
//+------------------------------------------------------------------+
bool ArrowedLinePointChange(const long   chart_ID=0,          // 图表 ID
                            const string name="ArrowedLine", // 线的名称
                            const int    point_index=0,       // 定位点指数
                            datetime     time=0,              // 定位点时间坐标
                            double       price=0)             // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动线的定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数移除图表的箭头线                                           |
//+------------------------------------------------------------------+
bool ArrowedLineDelete(const long   chart_ID=0,          // 图表 ID
                       const string name="ArrowedLine") // 线的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除箭头线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to create an arrowed line! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                         |
//| 默认的值                                                          |
//+------------------------------------------------------------------+
void ChangeArrowedLineEmptyPoints(datetime &time1,double &price1,
```

© 2000-2025, MetaQuotes Ltd.

```
                                    datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二点的价格没有设置，则它与第一点的价格相等
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变线定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义画线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建箭头线
   if(!ArrowedLineCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动线的定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 垂直移动第二个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p2<accuracy-1)
         p2+=1;
      //--- 移动点
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 垂直移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
```

© 2000-2025, MetaQuotes Ltd.

```
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 半秒延迟
   Sleep(500);
//--- 循环计数器
   int h_steps=bars/2;
//--- 同时水平移动两个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- 切换点
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.03 秒延迟
      Sleep(30);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除箭头线
   ArrowedLineDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_CHANNEL

等距通道

注意

对于等距通道，它可以指定延续向右和/或向左的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 根据属性）。也可以设置颜色填充通道的模式。

示例

下面的脚本创建和移动图表上的等距通道。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Equidistant Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Channel";   // 通道名称
input int              InpDate1=25;         // 第1个点的日期，%
input int              InpPrice1=60;        // 第1个点的价格，%
input int              InpDate2=65;         // 第2个点的日期，%
input int              InpPrice2=80;        // 第2个点的价格，%
input int              InpDate3=30;         // 第3个点的日期，%
input int              InpPrice3=40;        // 第3个点的价格，%
input color            InpColor=clrRed;     // 通道的颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // 通道线的风格
input int             InpWidth=2;          // 通道线的宽度
input bool            InpBack=false;       // 背景通道
input bool            InpFill=false;       // 填充通道颜色
input bool            InpSelection=true;   // 突出移动
input bool            InpRayLeft=false;    // 通道延续向左
input bool            InpRayRight=false;   // 通道延续向右
input bool            InpHidden=true;      // 隐藏在对象列表
input long            InpZOrder=0;         // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建等距通道                                       |
//+------------------------------------------------------------------+
bool ChannelCreate(const long            chart_ID=0,        // 图表 ID
                   const string          name="Channel",    // 通道的名称
                   const int             sub_window=0,      // 子窗口指数
                   datetime              time1=0,           // 第一个点的时间
                   double                price1=0,          //第一个点的价格
                   datetime              time2=0,           // 第二个点的时间
                   double                price2=0,          // 第二个点的价格
                   datetime              time3=0,           //第三个点的时间
                   double                price3=0,          // 第三个点的价格
                   const color           clr=clrRed,        // 通道的颜色
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // 通道线的风格
                   const int             width=1,           // 通道线的宽度
                   const bool            fill=false,        // 填充通道颜色
                   const bool            back=false,        // 在背景中
                   const bool            selection=true,    // 突出移动
                   const bool            ray_left=false,    // 通道延续向左
                   const bool            ray_right=false,   // 通道延续向右
                   const bool            hidden=true,       // 隐藏在对象列表
                   const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建通道
   if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time2,price2,tir
     {
      Print(__FUNCTION__,
            ": failed to create an equidistant channel! Error code = ",GetLastError())
      return(false);
     }
//--- 设置通道颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置通道线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置通道线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 启用（true）或禁用（false）填充通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用（true）或禁用（false）延续向左显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用（true）或禁用（false）延续向右显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动通道的定位点                                                 |
//+------------------------------------------------------------------+
bool ChannelPointChange(const long   chart_ID=0,      // 图表 ID
                        const string name="Channel", // 通道的名称
                        const int    point_index=0,  // 定位点指数
                        datetime     time=0,         // 定位点时间坐标
                        double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除通道                                                         |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
bool ChannelDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="Channel") // 通道的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除通道
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------+
//| 检查通道定位点的值和为空点设置                                         |
//| 默认的值                                                               |
//+------------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- 如果第二点（右侧）的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点（左侧）的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则高于第二个点移动300点
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第三个点的时间没有设置，则它与第一个点的时间相一致
   if(!time3)
      time3=time1;
//--- 如果第三个点的价格没有设置，则它与第二点的价格相等
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 脚本程序起始函数                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变通道定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制通道的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建等距通道
   if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,
     {
```

© 2000-2025, MetaQuotes Ltd.

```
         return;
        }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动通道定位点
//--- 循环计数器
   int h_steps=bars/6;
//--- 移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      //--- 移动点
      if(!ChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 移动第一个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1>1)
         d1-=1;
      //--- 移动点
      if(!ChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int v_steps=accuracy/10;
//--- 移动第三个定位点
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p3>1)
         p3-=1;
      //--- 移动点
      if(!ChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表通道
   ChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_STDDEVCHANNEL

标准偏差通道。

注意

对标准偏差通道，它可以指定延续向右和/或向左展示的模式 (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT 相应的属性)。也可以设置颜色填充通道的模式。

OBJPROP_DEVIATION 属性用于改变通道偏差的值。

示例

下面的脚本创建和移动图表上的标准偏差通道。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Standard Deviation Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string          InpName="StdDevChannel";    // 通道名称
input int             InpDate1=10;                // 第1个点的日期，%
input int             InpDate2=40;                // 第2个点的日期，%
input double          InpDeviation=1.0;           // 偏差
input color           InpColor=clrRed;            // 通道的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 通道线的风格
input int             InpWidth=2;                 // 通道线的宽度
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpFill=false;              // 填充通道颜色
input bool              InpBack=false;              // 背景通道
input bool              InpSelection=true;          // 突出移动
input bool              InpRayLeft=false;           // 通道延续向左
input bool              InpRayRight=false;          // 通道延续向右
input bool              InpHidden=true;             // 隐藏在对象列表
input long              InpZOrder=0;                // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建标准偏差通道                                     |
//+------------------------------------------------------------------+
bool StdDevChannelCreate(const long             chart_ID=0,         // 图表 ID
                         const string           name="Channel",     // 通道的名称
                         const int              sub_window=0,       // 子窗口指数
                         datetime               time1=0,            // 第一个点的时间
                         datetime               time2=0,            // 第二个点的时间
                         const double           deviation=1.0,      // 偏差
                         const color            clr=clrRed,         // 通道颜色
                         const ENUM_LINE_STYLE  style=STYLE_SOLID,  // 通道线的风格
                         const int              width=1,            // 通道线的宽度
                         const bool             fill=false,         // 填充通道颜色
                         const bool             back=false,         // 在背景中
                         const bool             selection=true,     // 突出移动
                         const bool             ray_left=false,     // 通道延续向左
                         const bool             ray_right=false,    // 通道延续向右
                         const bool             hidden=true,        // 隐藏在对象列表
                         const long             z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeChannelEmptyPoints(time1,time2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建通道
   if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create standard deviation channel! Error code = ",GetLastErro
      return(false);
     }
//--- 设置影响通道宽度的偏差值
   ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
//--- 设置通道颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置通道线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置通道线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 启用 (true) 或禁用 (false) 填充通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动通道的定位点                                                 |
//+------------------------------------------------------------------+
bool StdDevChannelPointChange(const long   chart_ID=0,      // 图表 ID
                              const string name="Channel", // 通道的名称
                              const int    point_index=0,  // 定位点指数
                              datetime     time=0)         // 定位点时间坐标
  {
//--- 如果没有设置点的时间，将点移动到当前柱
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 更改通道的偏差                                                   |
//+------------------------------------------------------------------+
bool StdDevChannelDeviationChange(const long   chart_ID=0,      // 图表 ID
                                  const string name="Channel", // 通道名称
                                  const double deviation=1.0)  // 偏差
  {
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 改变趋势线的斜角
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
     {
      Print(__FUNCTION__,
            ": failed to change channel deviation! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除通道                                                         |
//+------------------------------------------------------------------+
bool StdDevChannelDelete(const long   chart_ID=0,     // 图表 ID
                         const string name="Channel") // 通道的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除通道
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------+
//| 检查通道定位点的值和为空点设置                                         |
//| 默认的值                                                               |
//+------------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 脚本程序起始函数                                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变通道定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制通道的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- 创建标准偏差通道。
   if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpColor,InpStyl
      InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 现在，通道水平向右延展
//--- 循环计数器
   int h_steps=bars/2;
//--- 移动通道
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- 移动定位点
      if(!StdDevChannelPointChange(0,InpName,0,date[d1]))
         return;
      if(!StdDevChannelPointChange(0,InpName,1,date[d2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   double v_steps=InpDeviation*2;
//--- 扩展通道
   for(double i=InpDeviation;i<v_steps;i+=10.0/accuracy)
     {
      if(!StdDevChannelDeviationChange(0,InpName,i))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表通道
   StdDevChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_REGRESSION

线性回归通道

注意

对线性回归通道，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性）。也可以设置颜色填充通道的模式。

示例

下面的脚本创建和移动图表上的线性回归通道。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Linear Regression Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Regression"; // 通道名称
input int              InpDate1=10;          // 第1个点的日期，%
input int              InpDate2=40;          // 第2个点的日期，%
input color            InpColor=clrRed;      // 通道的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // 通道线的风格
input int              InpWidth=2;           // 通道线的宽度
input bool             InpFill=false;        // 填充通道颜色
input bool             InpBack=false;        // 背景通道
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpSelection=true;     // 突出移动
input bool              InpRayLeft=false;      // 通道延续向左
input bool              InpRayRight=false;     // 通道延续向右
input bool              InpHidden=true;        // 隐藏在对象列表
input long              InpZOrder=0;           // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建线性回归通道                                   |
//+------------------------------------------------------------------+
bool RegressionCreate(const long            chart_ID=0,         // 图表 ID
                      const string          name="Regression", // 通道名称
                      const int             sub_window=0,       // 子窗口指数
                      datetime              time1=0,            // 第一个点的时间
                      datetime              time2=0,            // 第二个点的时间
                      const color           clr=clrRed,         // 通道的颜色
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // 通道线的风格
                      const int             width=1,            // 通道线的宽度
                      const bool            fill=false,         // 填充通道颜色
                      const bool            back=false,         // 在背景中
                      const bool            selection=true,     // 突出移动
                      const bool            ray_left=false,     // 通道延续到向左
                      const bool            ray_right=false,    // 通道延续到向右
                      const bool            hidden=true,        // 隐藏在对象列表
                      const long            z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeRegressionEmptyPoints(time1,time2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建通道
   if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create linear regression channel! Error code = ",GetLastError
      return(false);
     }
//--- 设置通道颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置通道线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置通道线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 启用 (true) 或禁用 (false) 填充通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动通道的定位点                                                 |
//+------------------------------------------------------------------+
bool RegressionPointChange(const long   chart_ID=0,      //图表 ID
                           const string name="Channel", // 通道名称
                           const int    point_index=0,  // 定位点指数
                           datetime     time=0)         // 定位点时间坐标
  {
//--- 如果没有设置点的时间，将点移动到当前柱
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除通道                                                         |
//+------------------------------------------------------------------+
bool RegressionDelete(const long   chart_ID=0,     // 图表 ID
                      const string name="Channel") // 通道名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除通道
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 检查通道定位点的值和为空点设置                                                 |
//| 默认的值                                                                     |
//+------------------------------------------------------------------------------+
void ChangeRegressionEmptyPoints(datetime &time1,datetime &time2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变通道定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
```

© 2000-2025, MetaQuotes Ltd.

```
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制通道的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- 创建线性回归通道
   if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,InpWidth,
      InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，水平向右移动通道
//--- 循环计数器
   int h_steps=bars/2;
//--- 移动通道
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- 移动定位点
      if(!RegressionPointChange(0,InpName,0,date[d1]))
         return;
      if(!RegressionPointChange(0,InpName,1,date[d2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表通道
   RegressionDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_PITCHFORK

安德鲁鱼叉线。

注意

对于安德鲁鱼叉线，它可以指定延续向右和/或向左展示的模式(OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性)。

您也可以指定线的级别数，它们的值和颜色。

示例

下面的脚本创建和移动图表上的安德鲁鱼叉线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Andrews' Pitchfork\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Pitchfork";  // 鱼叉名称
input int              InpDate1=14;          // 第1个点的日期，%
input int              InpPrice1=40;         // 第1个点的价格，%
input int              InpDate2=18;          // 第2个点的日期，%
input int              InpPrice2=50;         // 第2个点的价格，%
input int              InpDate3=18;          // 第3个点的日期，%
input int              InpPrice3=30;         // 第3个点的价格，%
input color            InpColor=clrRed;      // 鱼叉颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // 鱼叉线的风格
input int             InpWidth=1;           // 鱼叉线的宽度
input bool            InpBack=false;        // 背景鱼叉线
input bool            InpSelection=true;    // 突出移动
input bool            InpRayLeft=false;     // 鱼叉线延续向左
input bool            InpRayRight=false;    // 鱼叉线延续向右
input bool            InpHidden=true;       // 隐藏在对象列表
input long            InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建安德鲁鱼叉线                                   |
//+------------------------------------------------------------------+
bool PitchforkCreate(const long            chart_ID=0,        // 图表 ID
                     const string          name="Pitchfork",  // 鱼叉线名称
                     const int             sub_window=0,      // 子窗口指数
                     datetime              time1=0,           // 第一个点的时间
                     double                price1=0,          // 第一个点的价格
                     datetime              time2=0,           // 第二个点的时间
                     double                price2=0,          // 第二个点的价格
                     datetime              time3=0,           // 第三个点的时间
                     double                price3=0,          // 第三个点的价格
                     const color           clr=clrRed,        // 鱼叉线的颜色
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // 鱼叉线的风格
                     const int             width=1,           // 鱼叉线的宽度
                     const bool            back=false,        // 在背景中
                     const bool            selection=true,    // 突出移动
                     const bool            ray_left=false,    // 鱼叉线延续向左
                     const bool            ray_right=false,   // 鱼叉线延续向右
                     const bool            hidden=true,       // 隐藏在对象列表
                     const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建安德鲁鱼叉线
   if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create \"Andrews' Pitchfork\"! Error code = ",GetLastError())
      return(false);
     }
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//---设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 启用 (true) 或禁用 (false) 突出鱼叉线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示鱼叉线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示鱼叉线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置安德鲁鱼叉线水平的数量和参数                                 |
//+------------------------------------------------------------------+
bool PitchforkLevelsSet(int              levels,           // 水平线的数量
                        double           &values[],        // 水平线的值
                        color            &colors[],        // 水平线的颜色
                        ENUM_LINE_STYLE  &styles[],        // 水平线的风格
                        int              &widths[],        // 水平线的宽度
                        const long       chart_ID=0,       // 图表 ID
                        const string     name="Pitchfork") // 鱼叉线的名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- 水平的描述
```

© 2000-2025, MetaQuotes Ltd.

```
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动安德鲁鱼叉线的定位点                                         |
//+------------------------------------------------------------------+
bool PitchforkPointChange(const long   chart_ID=0,        // 图表 ID
                          const string name="Pitchfork", // 通道的名称
                          const int    point_index=0,    // 定位点指数
                          datetime     time=0,           // 定位点时间坐标
                          double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除安德鲁鱼叉线                                                 |
//+------------------------------------------------------------------+
bool PitchforkDelete(const long   chart_ID=0,       // 图表 ID
                     const string name="Pitchfork") // 通道的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除通道
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Andrews' Pitchfork\"! Error code = ",GetLastError())
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 检查安德鲁鱼叉线定位点的值和为空点设置                                |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- 如果第二点（右上侧）的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点（左侧）的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则低于第二个点移动200点
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第三个点的时间没有设置，则它与第二个点的时间相一致
   if(!time3)
      time3=time2;
//---如果第三个点的价格没有设置，则低于第一个点移动200点
   if(!price3)
      price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变安德鲁鱼叉线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制安德鲁鱼叉线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建安德鲁鱼叉线
   if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pric
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidder
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动鱼叉线定位点
//--- 循环计数器
   int v_steps=accuracy/10;
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!PitchforkPointChange(0,InpName,0,date[d1],price[p1]))
```

© 2000-2025, MetaQuotes Ltd.

```
         return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
         return;
       //--- 重画图表
       ChartRedraw();
      }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars/8;
//--- 移动第三个定位点
   for(int i=0;i<h_steps;i++)
     {
       //--- 使用下面的值
       if(d3<bars-1)
         d3+=1;
       //--- 移动点
       if(!PitchforkPointChange(0,InpName,2,date[d3],price[p3]))
         return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
         return;
       //--- 重画图表
       ChartRedraw();
       //--- 重画图表
       ChartRedraw();
       // 0.05 秒延迟
       Sleep(50);
      }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy/10;
//--- 移动第二个定位点
   for(int i=0;i<v_steps;i++)
     {
       //--- 使用下面的值
       if(p2>1)
         p2-=1;
       //--- 移动点
       if(!PitchforkPointChange(0,InpName,1,date[d2],price[p2]))
         return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
         return;
       //--- 重画图表
       ChartRedraw();
      }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表鱼叉线
   PitchforkDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_GANNLINE

江恩线。

注意

对于江恩线，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性）。

江恩比例角度线和第二定位点的坐标都可以用于设置线的倾斜度。

示例

下面的脚本创建和移动图表上的江恩线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Gann Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="GannLine";         // 线的名称
input int              InpDate1=20;                // 第1个点的日期，%
input int              InpPrice1=75;               // 第1个点的价格， %
input int              InpDate2=80;                // 第2个点的日期，%
input double           InpAngle=0.0;               // 江恩角度线
input double           InpScale=1.0;               // 比例
input color            InpColor=clrRed;            // 线的颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 线的风格
input int             InpWidth=2;                // 线的宽度
input bool            InpBack=false;             // 背景线
input bool            InpSelection=true;         // 突出移动
input bool            InpRayLeft=false;          // 线延续向左
input bool            InpRayRight=true;          // 线延续向右
input bool            InpHidden=true;            // 隐藏在对象列表
input long            InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标，角度和比例创建江恩线                                |
//+------------------------------------------------------------------+
bool GannLineCreate(const long            chart_ID=0,        // 图表 ID
                    const string          name="GannLine",   // 线的名称
                    const int             sub_window=0,      // 子窗口指数
                    datetime              time1=0,           // 第一个点的时间
                    double                price1=0,          // 第一个点的价格
                    datetime              time2=0,           // 第二个点的时间
                    const double          angle=1.0,         // 江恩角度线
                    const double          scale=1.0,         // 比例
                    const color           clr=clrRed,        // 线的颜色
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // 线的风格
                    const int             width=1,           // 线的宽度
                    const bool            back=false,        // 在背景中
                    const bool            selection=true,    // 突出移动
                    const bool            ray_left=false,    // 线延续向左
                    const bool            ray_right=true,    // 线延续向右
                    const bool            hidden=true,       // 隐藏在对象列表
                    const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeGannLineEmptyPoints(time1,price1,time2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建江恩线
//--- 第二定位点的正确坐标被重新定义
//--- 江恩角度线和/或比例自动更改后，
   if(!ObjectCreate(chart_ID,name,OBJ_GANNLINE,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Line\"! Error code = ",GetLastError());
      return(false);
     }
//--- 改变江恩角度线
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- 改变比例 (每柱的点数)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- 设置线的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的显示风格
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用（true）或禁用（false）延续向左显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用（true）或禁用（false）延续向右显示线的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true）或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动江恩线定位点                                                 |
//+------------------------------------------------------------------+
bool GannLinePointChange(const long   chart_ID=0,       // 图表 ID
                         const string name="GannLine", // 线的名称
                         const int    point_index=0,   // 定位点指数
                         datetime     time=0,          // 定位点时间坐标
                         double       price=0)         // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动线的定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 改变江恩角度线                                                   |
//+------------------------------------------------------------------+
bool GannLineAngleChange(const long   chart_ID=0,       // 图表 ID
                         const string name="GannLine", // 线的名称
                         const double angle=1.0)       // 江恩角度线
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变江恩角度线
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change Gann angle! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变江恩线的比例                                                 |
//+------------------------------------------------------------------+
bool GannLineScaleChange(const long   chart_ID=0,       // 图表 ID
                         const string name="GannLine", // 线的名称
                         const double scale=1.0)       // 比例
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变比例 (每柱的点数)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数移除图表江恩线                                           |
//+------------------------------------------------------------------+
bool GannLineDelete(const long   chart_ID=0,       // 图表 ID
                    const string name="GannLine") // 线的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除江恩线。
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
```

© 2000-2025, MetaQuotes Ltd.

```
            ": failed to delete \"Gann Line\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查江恩线定位点的值和为空点设置                                 |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变线定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
```

© 2000-2025, MetaQuotes Ltd.

```
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制江恩线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- 创建江恩线
   if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,InpScale,InpCol
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrde
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动线的定位点和改变角度
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 垂直移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 半秒延迟
```

© 2000-2025, MetaQuotes Ltd.

```
   Sleep(500);
//--- 定义江恩角度线的当前值（已变化
//--- 当移动第一个定位点以后）
   double curr_angle;
   if(!ObjectGetDouble(0,InpName,OBJPROP_ANGLE,0,curr_angle))
      return;
//--- 循环计数器
   v_steps=accuracy/8;
//--- 改变江恩角度线
   for(int i=0;i<v_steps;i++)
     {
      if(!GannLineAngleChange(0,InpName,curr_angle-0.05*i))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表线
   GannLineDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_GANNFAN

江恩扇形线。

注意

对于江恩扇形线，可以指定 ENUM_GANN_DIRECTION 枚举的趋势类型。通过调整比例值 (OBJPROP_SCALE)，可以改变扇形线的倾斜角。

示例

下面的脚本创建和移动图表上的江恩扇形线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Gann Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string          InpName="GannFan";          // 扇形线名称
input int             InpDate1=15;                // 第1个点的日期，%
input int             InpPrice1=25;               // 第1个点的价格，%
input int             InpDate2=85;                // 第2个点的日期，%
input double          InpScale=2.0;               // 比例
input bool            InpDirection=false;         // 趋势方向
input color           InpColor=clrRed;            // 扇形线的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 扇形线的风格
```

© 2000-2025, MetaQuotes Ltd.

```
input int              InpWidth=1;                // 扇形线的宽度
input bool             InpBack=false;             // 背景扇形线
input bool             InpSelection=true;         // 突出移动
input bool             InpHidden=true;            // 隐藏在对象列表
input long             InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建江恩扇形线                                                   |
//+------------------------------------------------------------------+
bool GannFanCreate(const long            chart_ID=0,        // 图表 ID
                   const string          name="GannFan",    // 扇形线的名称
                   const int             sub_window=0,      // 子窗口指数
                   datetime              time1=0,           // 第一个点的时间
                   double                price1=0,          //第一个点的价格
                   datetime              time2=0,           // 第二个点的时间
                   const double          scale=1.0,         // 比例
                   const bool            direction=true,    // 趋势方向
                   const color           clr=clrRed,        // 扇形线颜色
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // 扇形线的风格
                   const int             width=1,           // 扇形线的宽度
                   const bool            back=false,        // 在背景中
                   const bool            selection=true,    // 突出移动
                   const bool            hidden=true,       // 隐藏在对象列表
                   const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeGannFanEmptyPoints(time1,price1,time2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建江恩扇形线
   if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- 改变比例（每柱的点数）
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- 改变江恩扇形线趋势方向（true - 下降，false - 上升）
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- 设置扇形线颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置扇形线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置扇形线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出扇形线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动江恩扇形线定位点                                             |
//+------------------------------------------------------------------+
bool GannFanPointChange(const long   chart_ID=0,      // 图表 ID
                        const string name="GannFan", // 扇形线的名称
                        const int    point_index=0,  // 定位点指数
                        datetime     time=0,         // 定位点时间坐标
                        double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动扇形线的定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变江恩扇形线的比例                                             |
//+------------------------------------------------------------------+
bool GannFanScaleChange(const long   chart_ID=0,      // 图表 ID
                        const string name="GannFan", // 扇形线的名称
                        const double scale=1.0)      // scale
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变比例 (每柱的点数)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变江恩扇形线的趋势方向                                         |
//+------------------------------------------------------------------+
bool GannFanDirectionChange(const long   chart_ID=0,     // 图表 ID
                            const string name="GannFan", // 扇形线的名称
                            const bool   direction=true) // 趋势方向
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变江恩扇形线的趋势方向
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数移除图表的江恩扇形线                                     |
//+------------------------------------------------------------------+
bool GannFanDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="GannFan") // 扇形线的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除江恩扇形线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查江恩扇形线定位点的值和为空点设置                             |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &time2)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变扇形线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
```

© 2000-2025, MetaQuotes Ltd.

```
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制江恩扇形线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- 创建江恩扇形线
   if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动扇形线的定位点
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 垂直移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1<accuracy-1)
         p1+=1;
      //--- 移动点
      if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 改变扇形线的趋势方向降一级
   GannFanDirectionChange(0,InpName,true);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表扇形线
   GannFanDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
```

© 2000-2025, MetaQuotes Ltd.

```
    Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_GANNGRID

江恩网格。

注意

对于江恩网格，可以从 ENUM_GANN_DIRECTION指定趋势类型。通过调整比例值(OBJPROP_SCALE)，可以改变网格线的倾斜角。

示例

下面的脚本创建和移动图表上的江恩网格。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Gann Grid\" graphical object."
#property description "Anchor point coordinates of the grid are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string          InpName="GannGrid";         // 网格名称
input int             InpDate1=15;                // 第1个点的日期，%
input int             InpPrice1=25;               // 第1个点的价格，%
input int             InpDate2=35;                // 第2个点的日期，%
input double          InpScale=3.0;               // 比例
input bool            InpDirection=false;         // 趋势方向
input color           InpColor=clrRed;            // 网格颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 网格线的风格
```

© 2000-2025, MetaQuotes Ltd.

```
input int              InpWidth=1;                // 扇形线的宽度
input bool             InpBack=false;             // 背景网格
input bool             InpSelection=true;         // 突出移动
input bool             InpHidden=true;            // 隐藏在对象列表
input long             InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建江恩网格                                                     |
//+------------------------------------------------------------------+
bool GannGridCreate(const long             chart_ID=0,        // 图表ID
                    const string           name="GannGrid",   // 网格的名称
                    const int              sub_window=0,      // 子窗口指数
                    datetime               time1=0,           // 第一个点的时间
                    double                 price1=0,          // 第一个点的价格
                    datetime               time2=0,           // 第二个点的时间
                    const double           scale=1.0,         // 比例
                    const bool             direction=true,    // 趋势方向
                    const color            clr=clrRed,        // 网格颜色
                    const ENUM_LINE_STYLE  style=STYLE_SOLID, // 网格线的风格
                    const int              width=1,           // 网格线的宽度
                    const bool             back=false,        // 在背景中
                    const bool             selection=true,    // 突出移动
                    const bool             hidden=true,       // 隐藏在对象列表
                    const long             z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeGannGridEmptyPoints(time1,price1,time2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建江恩网格
   if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Grid\"! Error code = ",GetLastError());
      return(false);
     }
//--- 改变比例（每柱的点数）
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- 改变江恩扇形线趋势方向（true - 下降，false - 上升）
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- 设置网格颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置网格线的显示风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置网格线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出网格线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动江恩网格定位点                                                 |
//+------------------------------------------------------------------+
bool GannGridPointChange(const long   chart_ID=0,       // 图表 ID
                         const string name="GannGrid", // 网格的名称
                         const int    point_index=0,   // 定位点指数
                         datetime     time=0,          // 定位点时间坐标
                         double       price=0)         // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动网格的定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变江恩网格的比例                                                 |
//+------------------------------------------------------------------+
bool GannGridScaleChange(const long   chart_ID=0,       // 图表 ID
                         const string name="GannGrid", // 网格
                         const double scale=1.0)       // 比例
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变比例 (每柱的点数)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变江恩网格线的趋势方向                                         |
//+------------------------------------------------------------------+
bool GannGridDirectionChange(const long   chart_ID=0,      // 图表 ID
                             const string name="GannGrid", // 网格的名称
                             const bool   direction=true)  // 趋势方向
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变江恩网格线的趋势方向
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 这个函数移除图表的江恩扇形线                                     |
//+------------------------------------------------------------------+
bool GannGridDelete(const long   chart_ID=0,      // 图表 ID
                    const string name="GannGrid") // 网格的名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除江恩网格
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Grid\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查江恩网格线定位点的值和为空点设置                             |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &time2)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变网格定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
```

© 2000-2025, MetaQuotes Ltd.

```
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制江恩网格线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- 创建江恩网格
   if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- now, move the grid's anchor points
//--- 循环计数器
   int v_steps=accuracy/4;
//--- 垂直移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1<accuracy-1)
         p1+=1;
      if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars/4;
//---水平移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      if(!GannGridPointChange(0,InpName,1,date[d2],0))
         return;
      //--- 检查脚本操作是否已经强制禁用
```

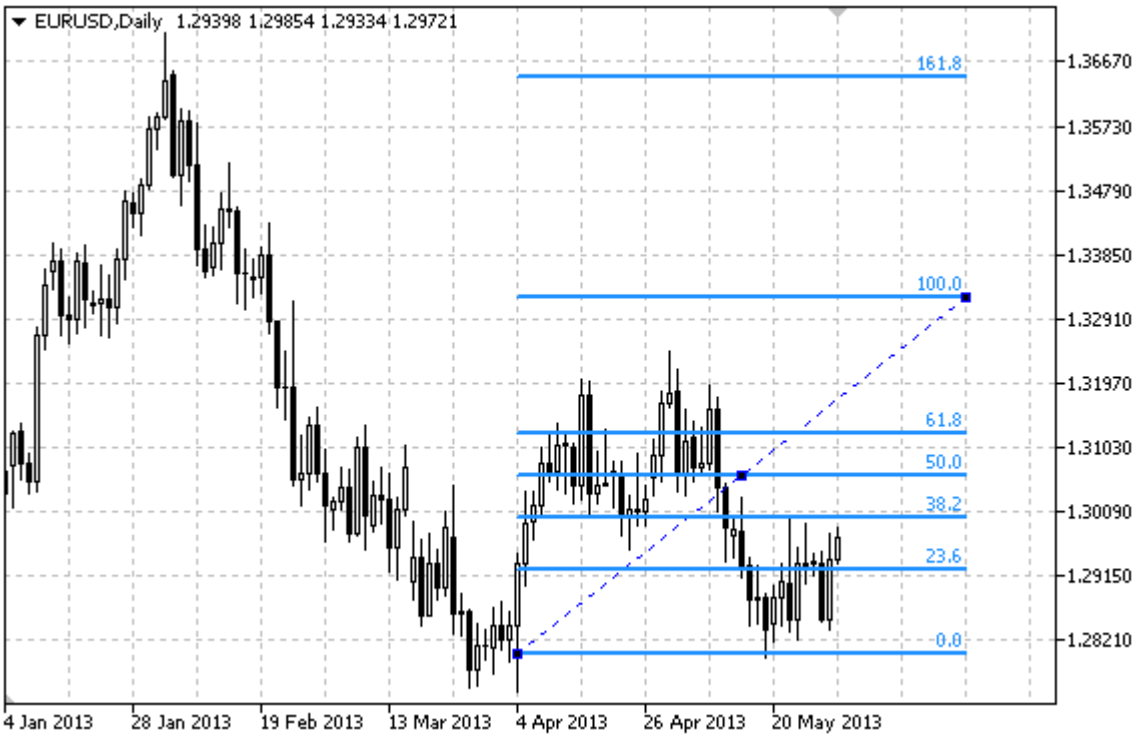
© 2000-2025, MetaQuotes Ltd.

```
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 改变网格线的趋势方向降一级
   GannGridDirectionChange(0,InpName,true);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表网格
   GannGridDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_FIBO

斐波纳契回调线




注意

对于斐波纳契回调线，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性）。

您也可以指定线的级别数，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契回调线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Retracement\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string          InpName="FiboLevels";       // 对象名称
input int             InpDate1=10;                // 第1个点的日期，%
input int             InpPrice1=65;               // 第1个点的价格，%
input int             InpDate2=90;                // 第2个点的日期，%
input int             InpPrice2=85;               // 第2个点的价格，%
input color           InpColor=clrRed;            // 对象的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 线的风格
```

© 2000-2025, MetaQuotes Ltd.

```
input int              InpWidth=2;                // 线的宽度
input bool             InpBack=false;             // 背景对象
input bool             InpSelection=true;         // 突出移动
input bool             InpRayLeft=false;          // 对象延续向左
input bool             InpRayRight=false;         // 对象延续向右
input bool             InpHidden=true;            // 隐藏在对象列表
input long             InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契回调线                                 |
//+------------------------------------------------------------------+
bool FiboLevelsCreate(const long            chart_ID=0,        // 图表 ID
                      const string          name="FiboLevels", // 对象名称
                      const int             sub_window=0,      // 子窗口指数
                      datetime              time1=0,           // 第一个点的时间
                      double                price1=0,          // 第一个点的价格
                      datetime              time2=0,           // 第二个点的时间
                      double                price2=0,          // 第二个点的价格
                      const color           clr=clrRed,        // 对象颜色
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // 对象线的风格
                      const int             width=1,           // 对象线的宽度
                      const bool            back=false,        // 在背景中
                      const bool            selection=true,    // 突出移动
                      const bool            ray_left=false,    // 对象持续向左
                      const bool            ray_right=false,   // 对象持续向右
                      const bool            hidden=true,       // 隐藏在对象列表
                      const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 用给定的坐标创建斐波纳契回调线
   if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Retracement\"! Error code = ",GetLastError
      return(false);
     }
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
```

© 2000-2025, MetaQuotes Ltd.

```
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示对象的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示对象的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                             |
//+------------------------------------------------------------------+
bool FiboLevelsSet(int             levels,           // 水平线的数量
                   double          &values[],        // 水平线的值
                   color           &colors[],        // 水平线的颜色
                   ENUM_LINE_STYLE &styles[],        // 水平线的风格
                   int             &widths[],        // 水平线的宽度
                   const long      chart_ID=0,       // 图表ID
                   const string    name="FiboLevels") // 对象名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) || levels!=ArraySize(widt
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契回调线的定位点                                       |
//+------------------------------------------------------------------+
bool FiboLevelsPointChange(const long   chart_ID=0,         // 图表 ID
                           const string name="FiboLevels", // 对象名称
                           const int    point_index=0,     // 定位点指数
                           datetime     time=0,            // 定位点时间坐标
                           double       price=0)           // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除斐波纳契回调线                                               |
//+------------------------------------------------------------------+
bool FiboLevelsDelete(const long   chart_ID=0,         // 图表 ID
                      const string name="FiboLevels") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Retracement\"! Error code = ",GetLastErro
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契回调线定位点的值和为空点设置                         |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                 datetime &time2,double &price2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则低于第二个点移动200点
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变斐波纳契回调线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制斐波纳契回调线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建对象
   if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrde
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy*2/5;
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy*4/5;
//--- 移动第二个定位点
   for(int i=0;i<v_steps;i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 使用下面的值
       if(p2>1)
          p2-=1;
       //--- 移动点
       if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
          return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
          return;
       //--- 重画图表
       ChartRedraw();
      }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   FiboLevelsDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_FIBOTIMES

斐波纳契时间周期线。

注意

对于"斐波纳契时间周期线"，可以指定线水平的数量，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契时间周期线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Time Zones\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="FiboTimes";       // 对象名称
input int              InpDate1=10;               // 第1个点的日期，%
input int              InpPrice1=45;              // 第1个点的价格，%
input int              InpDate2=20;               // 第2个点的日期，%
input int              InpPrice2=55;              // 第2个点的价格，%
input color            InpColor=clrRed;           // 对象的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT; // 线的风格
input int              InpWidth=2;                // 线的宽度
input bool             InpBack=false;             // 背景对象
input bool             InpSelection=true;         // 突出移动
```

© 2000-2025, MetaQuotes Ltd.

```
input bool            InpHidden=true;            // 隐藏在对象列表
input long            InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契时间周期线                                |
//+------------------------------------------------------------------+
bool FiboTimesCreate(const long            chart_ID=0,        // 图表 ID
                     const string          name="FiboTimes",  // 对象名称
                     const int             sub_window=0,      // 子窗口指数
                     datetime              time1=0,           // 第一个点的时间
                     double                price1=0,          // 第一个点的价格
                     datetime              time2=0,           // 第二个点的时间
                     double                price2=0,          // 第二个点的价格
                     const color           clr=clrRed,        // 对象颜色
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // 对象线的风格
                     const int             width=1,           // 对象线的宽度
                     const bool            back=false,        // 在背景中
                     const bool            selection=true,    // 突出移动
                     const bool            hidden=true,       // 隐藏在对象列表
                     const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboTimesEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建斐波纳契时间周期线
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Time Zones\"! Error code = ",GetLastError
      return(false);
     }
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                               |
//+------------------------------------------------------------------+
bool FiboTimesLevelsSet(int             levels,           // 水平线的数量
                        double          &values[],        // 水平线的值
                        color           &colors[],        // 水平线的颜色
                        ENUM_LINE_STYLE &styles[],        // 水平线的风格
                        int             &widths[],        // 水平线的宽度
                        const long      chart_ID=0,       // 图表 ID
                        const string    name="FiboTimes") // 对象名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(values[i],1));
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契时间周期线的定位点                                     |
//+------------------------------------------------------------------+
bool FiboTimesPointChange(const long   chart_ID=0,       // 图表 ID
                          const string name="FiboTimes", // 对象名称
                          const int    point_index=0,    // 定位点指数
                          datetime     time=0,           // 定位点时间坐标
                          double       price=0)          // 定位点价格坐标
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除斐波纳契时间周期线                                           |
//+------------------------------------------------------------------+
bool FiboTimesDelete(const long   chart_ID=0,       // 图表 ID
                     const string name="FiboTimes") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Time Zones\"! Error code = ",GetLastError
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契时间周期线定位点的值和                               |
//| 为空点设置默认的值                                               |
//+------------------------------------------------------------------+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的2个柱
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!time2)
     {
      //--- 接收最近3柱开盘时间的数组
      datetime temp[3];
      CopyTime(Symbol(),Period(),time1,3,temp);
      //--- 在第二点左侧2柱设置第一点
      time2=temp[0];
     }
//--- 如果第二点的价格没有设置，则它与第一点的价格相等
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变斐波纳契时间周期线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      price[i]=min_price+i*step;
//--- 定义绘制斐波纳契时间周期线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建对象
   if(!FiboTimesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int h_steps=bars*2/5;
//--- 移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      //--- 移动点
      if(!FiboTimesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   h_steps=bars*3/5;
//--- 移动第一个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      //--- 移动点
      if(!FiboTimesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
```

© 2000-2025, MetaQuotes Ltd.

```
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   FiboTimesDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_FIBOFAN

斐波纳契扇形线。

注意

对于"斐波纳契扇形线"，可以指定线水平的数量，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契扇形线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="FiboFan";          // 扇形线名称
input int              InpDate1=10;                // 第1个点的日期，%
input int              InpPrice1=25;               // 第1个点的价格，%
input int              InpDate2=30;                // 第2个点的日期，%
input int              InpPrice2=50;               // 第2个点的价格，%
input color            InpColor=clrRed;            // 扇形线的颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT; // 线的风格
input int              InpWidth=2;                 // 线的宽度
input bool             InpBack=false;              // 背景对象
```

© 2000-2025, MetaQuotes Ltd.

```
input bool             InpSelection=true;          // 突出移动
input bool             InpHidden=true;             // 隐藏在对象列表
input long             InpZOrder=0;                // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契扇形线                                  |
//+------------------------------------------------------------------+
bool FiboFanCreate(const long            chart_ID=0,         // 图表ID
                   const string          name="FiboFan",     // 扇形线名称
                   const int             sub_window=0,       // 子窗口指数
                   datetime              time1=0,            // 第一个点的时间
                   double                price1=0,           //第一个点的价格
                   datetime              time2=0,            // 第二个点的时间
                   double                price2=0,           // 第二个点的价格
                   const color           clr=clrRed,         // 扇形线颜色
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // 扇形线的风格
                   const int             width=1,            // 扇形线的宽度
                   const bool            back=false,         // 在背景中
                   const bool            selection=true,     // 突出移动
                   const bool            hidden=true,        // 隐藏在对象列表
                   const long            z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建斐波纳契扇形线
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出扇形线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                             |
//+------------------------------------------------------------------+
bool FiboFanLevelsSet(int             levels,          // 水平线的数量
                      double          &values[],       // 水平线的值
                      color           &colors[],       // 水平线的颜色
                      ENUM_LINE_STYLE &styles[],       // 水平线的风格
                      int             &widths[],       // 水平线的宽度
                      const long      chart_ID=0,      // 图表 ID
                      const string    name="FiboFan") // 扇形线名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契扇形线定位点                                         |
//+------------------------------------------------------------------+
bool FiboFanPointChange(const long   chart_ID=0,      // 图表 ID
                        const string name="FiboFan", // 扇形线名称
                        const int    point_index=0,  // 定位点指数
                        datetime     time=0,         // 定位点时间坐标
                        double       price=0)        // 定位点价格坐标
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除斐波纳契扇形线                                               |
//+------------------------------------------------------------------+
bool FiboFanDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="FiboFan") // 扇形线名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除扇形线
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契扇形线定位点的值和为空点设置                         |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则低于第二个点移动200点
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变斐波纳契扇形线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制斐波纳契扇形线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建对象
   if(!FiboFanCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动扇形线的定位点
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1<accuracy-1)
         p1+=1;
      //--- 移动点
      if(!FiboFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars/4;
//--- 移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      //--- 移动点
      if(!FiboFanPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
```

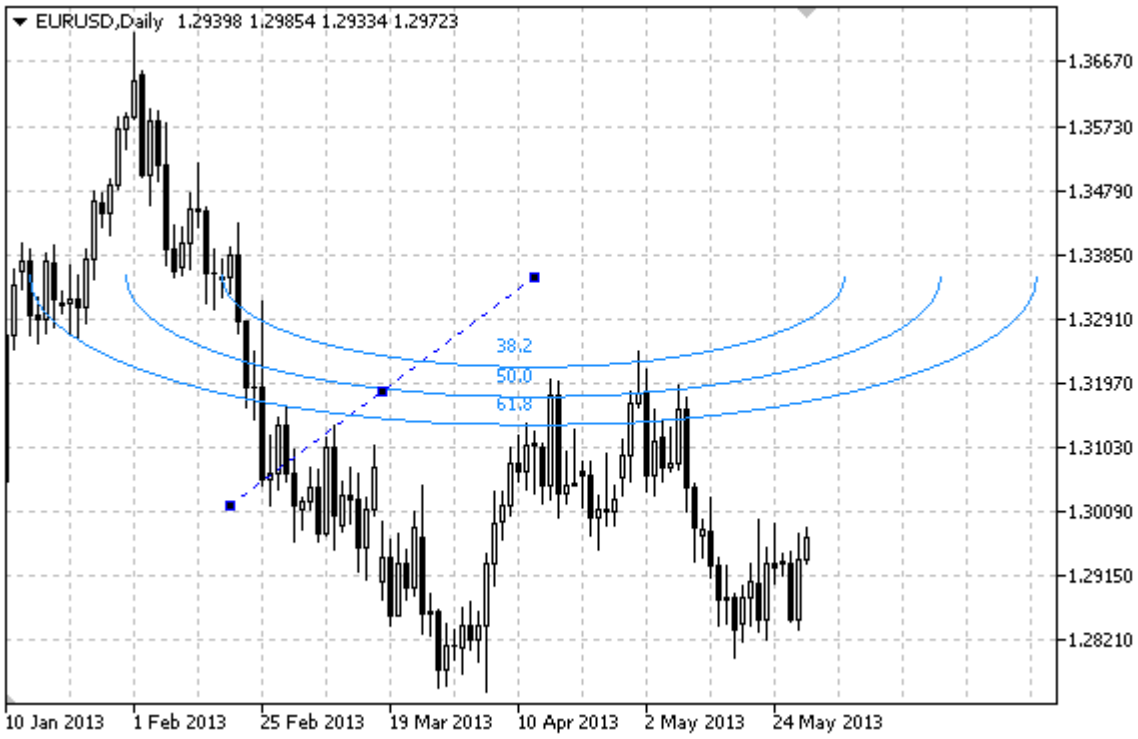
© 2000-2025, MetaQuotes Ltd.

```
            return;
         //--- 重画图表
         ChartRedraw();
         // 0.05 秒延迟
         Sleep(50);
        }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   FiboFanDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_FIBOARC

斐波纳契弧形线。




注意

对于 "斐波纳契弧形线"，可以指定整个椭圆的演示模式。曲率半径可以通过改变定位点比例和坐标来指定。

您也可以指定线的级别数，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契弧形线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Arcs\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="FiboArc";          // 对象名称
input int              InpDate1=25;                // 第1个点的日期，%
input int              InpPrice1=25;               // 第1个点的价格，%
input int              InpDate2=35;                // 第2个点的日期，%
input int              InpPrice2=55;               // 第2个点的价格，%
input double           InpScale=3.0;               // 比例
input bool             InpFullEllipse=true;        // 弧形线的形状
```

© 2000-2025, MetaQuotes Ltd.

```
input color           InpColor=clrRed;           // 线的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 线的风格
input int             InpWidth=2;                // 线的宽度
input bool            InpBack=false;             // 背景对象
input bool            InpSelection=true;         // 突出移动
input bool            InpHidden=true;            // 隐藏在对象列表
input long            InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契弧形线                                 |
//+------------------------------------------------------------------+
bool FiboArcCreate(const long            chart_ID=0,          // 图表 ID
                   const string          name="FiboArc",      // 对象名称
                   const int             sub_window=0,        // 子窗口指数
                   datetime              time1=0,             // 第一个点的时间
                   double                price1=0,            // 第一个点的价格
                   datetime              time2=0,             // 第二个点的时间
                   double                price2=0,            // 第二个点的价格
                   const double          scale=1.0,           // 比例
                   const bool            full_ellipse=false,  // 弧形线的形状
                   const color           clr=clrRed,          // 线的颜色
                   const ENUM_LINE_STYLE style=STYLE_SOLID,   // 线的风格
                   const int             width=1,             // 线的宽度
                   const bool            back=false,          // 在背景中
                   const bool            selection=true,      // 突出移动
                   const bool            hidden=true,         // 隐藏在对象列表
                   const long            z_order=0)           // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建斐波纳契弧形线
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Arcs\"! Error code = ",GetLastError());
      return(false);
     }
//--- 设置比例
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- 设置弧形线显示为全椭圆(true) 或半椭圆(false)
   ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出弧形线移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                             |
//+------------------------------------------------------------------+
bool FiboArcLevelsSet(int              levels,          // 水平线的数量
                      double           &values[],       // 水平线的值
                      color            &colors[],       // 水平线的颜色
                      ENUM_LINE_STYLE &styles[],       // 水平线的风格
                      int              &widths[],       // 水平线的宽度
                      const long       chart_ID=0,      // 图表 ID
                      const string     name="FiboArc") // 对象名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- 成功执行
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契弧形线的定位点                                       |
//+------------------------------------------------------------------+
bool FiboArcPointChange(const long   chart_ID=0,      // 图表 ID
                        const string name="FiboArc", // 对象名称
                        const int    point_index=0,  // 定位点指数
                        datetime     time=0,         // 定位点时间坐标
                        double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除斐波纳契弧形线                                               |
//+------------------------------------------------------------------+
bool FiboArcDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="FiboArc") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Arcs\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契弧形线定位点的值和为空点设置                         |
//| 默认的值                                                         |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- 如果第二点的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则低于第二个点移动300点
   if(!price1)
      price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变斐波纳契弧形线的定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Failed to copy time values! Error code = ",GetLastError());
       return;
      }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制斐波纳契弧形线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建对象
   if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpScale,
      InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrd
      {
       return;
      }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
      {
       //--- 使用下面的值
       if(p1<accuracy-1)
          p1+=1;
       //--- 移动点
       if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
          return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
          return;
       //--- 重画图表
       ChartRedraw();
      }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars/5;
//--- 移动第二个定位点
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2<bars-1)
         d2+=1;
      //--- 移动点
      if(!FiboArcPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   FiboArcDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_FIBOCHANNEL

斐波纳契通道线。

注意

对于斐波纳契通道线，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性）。

您也可以指定线的级别数，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契通道线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="FiboChannel";      // 通道名称
input int              InpDate1=20;                // 第1个点的日期，%
input int              InpPrice1=10;               // 第1个点的价格，%
input int              InpDate2=60;                // 第2个点的日期，%
input int              InpPrice2=30;               // 第2个点的价格，%
input int              InpDate3=20;                // 第3个点的日期，%
input int              InpPrice3=25;               // 第3个点的价格，%
```

© 2000-2025, MetaQuotes Ltd.

```
input color           InpColor=clrRed;           // 通道的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 通道线的风格
input int             InpWidth=2;                // 通道线的宽度
input bool            InpBack=false;             // 背景通道
input bool            InpSelection=true;         // 突出移动
input bool            InpRayLeft=false;          // 通道延续向左
input bool            InpRayRight=false;         // 通道延续向右
input bool            InpHidden=true;            // 隐藏在对象列表
input long            InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契通道线                                 |
//+------------------------------------------------------------------+
bool FiboChannelCreate(const long            chart_ID=0,          // 图表 ID
                       const string          name="FiboChannel", // 通道名称
                       const int             sub_window=0,       // 子窗口指数
                       datetime              time1=0,            // 第一个点的时间
                       double                price1=0,           // 第一个点的价格
                       datetime              time2=0,            // 第二个点的时间
                       double                price2=0,           // 第二个点的价格
                       datetime              time3=0,            // 第三个点的时间
                       double                price3=0,           // 第三个点的价格
                       const color           clr=clrRed,         // 通道的颜色
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // 通道线的风格
                       const int             width=1,            // 通道线的宽度
                       const bool            back=false,         // 在背景中
                       const bool            selection=true,     // 突出移动
                       const bool            ray_left=false,     // 通道延续向左
                       const bool            ray_right=false,    // 通道延续向右
                       const bool            hidden=true,        // 隐藏在对象列表
                       const long            z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建通道
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,time2,price2
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Channel\"! Error code = ",GetLastError());
      return(false);
     }
//--- 设置通道颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置通道线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置通道线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右显示通道的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                             |
//+------------------------------------------------------------------+
bool FiboChannelLevelsSet(int             levels,             // 水平线的数量
                          double          &values[],          // 水平线的值
                          color           &colors[],          // 水平线的颜色
                          ENUM_LINE_STYLE &styles[],          // 水平线的风格
                          int             &widths[],          // 水平线的宽度
                          const long      chart_ID=0,         // 图表 ID
                          const string    name="FiboChannel") // 对象名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契通道线的定位点                                       |
//+------------------------------------------------------------------+
bool FiboChannelPointChange(const long   chart_ID=0,          // 图表 ID
                            const string name="FiboChannel", // 通道名称
                            const int    point_index=0,      // 定位点指数
                            datetime     time=0,             // 定位点时间坐标
                            double       price=0)            // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除通道                                                         |
//+------------------------------------------------------------------+
bool FiboChannelDelete(const long   chart_ID=0,         // 图表 ID
                       const string name="FiboChannel") // 通道名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除通道
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Channel\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契通道定位点的值和为空点设置                                |
//| 默认的值                                                          |
//+------------------------------------------------------------------+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                  double &price2,datetime &time3,double &price3)
  {
//--- 如果第二点（右侧）的时间没有设置，它将位于当前柱
   if(!time2)
      time2=TimeCurrent();
//--- 如果第二点的价格没有设置，则它将用卖价值
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点（左侧）的时间没有设置，它则位于第二点左侧的9个柱
   if(!time1)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一个点的价格没有设置，则高于第二个点移动300点
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第三个点的时间没有设置，则它与第一个点的时间相一致
   if(!time3)
      time3=time1;
//--- 如果第三个点的价格没有设置，则它与第二点的价格相等
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
```

© 2000-2025, MetaQuotes Ltd.

```
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变通道定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制通道的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建斐波纳契通道
   if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pr
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动通道定位点
//--- 循环计数器
   int h_steps=bars/10;
//--- 移动第一个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1>1)
         d1-=1;
      //--- 移动点
```

© 2000-2025, MetaQuotes Ltd.

```
      if(!FiboChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int v_steps=accuracy/10;
//--- 移动第二个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p2>1)
         p2-=1;
      //--- 移动点
      if(!FiboChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy/15;
//--- 移动第三个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p3<accuracy-1)
         p3+=1;
      //--- 移动点
      if(!FiboChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
```

© 2000-2025, MetaQuotes Ltd.

```
   Sleep(1000);
//--- 删除图表通道
   FiboChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_EXPANSION

斐波纳契扩展线。

注意

对于"斐波纳契扩展线"，它可以指定延续向右和/或向左展示的模式（OBJPROP_RAY_RIGHT 和 OBJPROP_RAY_LEFT 相应的属性）。

您也可以指定线的级别数，它们的值和颜色。

示例

下面的脚本创建和移动图表上的斐波纳契扩展线。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Fibonacci Expansion\"graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string          InpName="FiboExpansion";   // 对象名称
input int             InpDate1=10;               // 第1个点的日期，%
input int             InpPrice1=55;              // 第1个点的价格，%
input int             InpDate2=30;               // 第2个点的日期，%
input int             InpPrice2=10;              // 第2个点的价格，%
input int             InpDate3=80;               // 第3个点的日期，%
input int             InpPrice3=75;              // 第3个点的价格，%
```

© 2000-2025, MetaQuotes Ltd.

```
input color           InpColor=clrRed;           // 对象的颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 线的风格
input int             InpWidth=2;                // 线的宽度
input bool            InpBack=false;             // 背景对象
input bool            InpSelection=true;         // 突出移动
input bool            InpRayLeft=false;          // 对象延续向左
input bool            InpRayRight=false;         // 对象延续向右
input bool            InpHidden=true;            // 隐藏在对象列表
input long            InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建斐波纳契扩展线                                 |
//+------------------------------------------------------------------+
bool FiboExpansionCreate(const long            chart_ID=0,           // 图表 ID
                         const string          name="FiboExpansion", // 通道的名称
                         const int             sub_window=0,         // 子窗口指数
                         datetime              time1=0,              // 第一个点的时间
                         double                price1=0,             // 第一个点的价格
                         datetime              time2=0,              // 第二个点的时间
                         double                price2=0,             // 第二个点的价格
                         datetime              time3=0,              // 第三个点的时间
                         double                price3=0,             // 第三个点的价格
                         const color           clr=clrRed,           // 对象颜色
                         const ENUM_LINE_STYLE style=STYLE_SOLID,    // 线的风格
                         const int             width=1,              // 线的宽度
                         const bool            back=false,           // 在背景中
                         const bool            selection=true,       // 突出移动
                         const bool            ray_left=false,       // 对象延续向左
                         const bool            ray_right=false,      // 对象延续向右
                         const bool            hidden=true,          // 隐藏在对象列表
                         const long            z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建斐波纳契扩展线
   if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Extension\"! Error code = ",GetLastError()
      return(false);
     }
//--- 设置对象的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//---设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 启用 (true) 或禁用 (false) 延续向左的对象可视化的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- 启用 (true) 或禁用 (false) 延续向右的对象可视化的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置水平的数量和参数                                             |
//+------------------------------------------------------------------+
bool FiboExpansionLevelsSet(int              levels,               // 水平线的数量
                            double           &values[],            // 水平线的值
                            color            &colors[],            // 水平线的颜色
                            ENUM_LINE_STYLE  &styles[],            // 水平线的风格
                            int              &widths[],            // 水平线的宽度
                            const long       chart_ID=0,           // 图表 ID
                            const string     name="FiboExpansion") // 对象名称
  {
//--- 检查数组大小
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- 设置水平数量
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- 设置循环中水平的属性
   for(int i=0;i<levels;i++)
     {
      //--- 水平的值
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- 水平的颜色
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- 水平的风格
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- 水平的宽度
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 水平的描述
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToString(100*value
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动斐波纳契扩展线的定位点                                       |
//+------------------------------------------------------------------+
bool FiboExpansionPointChange(const long   chart_ID=0,           // 图表 ID
                              const string name="FiboExpansion", // 对象名称
                              const int    point_index=0,        // 定位点指数
                              datetime     time=0,               // 定位点时间坐标
                              double       price=0)              // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除斐波纳契扩展线                                               |
//+------------------------------------------------------------------+
bool FiboExpansionDelete(const long   chart_ID=0,           // 图表 ID
                         const string name="FiboExpansion") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Expansion\"! Error code = ",GetLastError()
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 检查斐波纳契扩展线定位点的值和为空点设置                                  |
//| 默认的值                                                          |
//+------------------------------------------------------------------+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                    double &price2,datetime &time3,double &price3)
  {
//--- 如果第三点（右侧）的时间没有设置，它将位于当前柱
   if(!time3)
      time3=TimeCurrent();
//--- 如果第三点的价格没有设置，则它将用卖价值
   if(!price3)
      price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第一点（左侧）的时间没有设置，它则位于第三点左侧的9个柱
//--- 接收最近10柱开盘时间的数组
   datetime temp[];
   ArrayResize(temp,10);
   if(!time1)
     {
      CopyTime(Symbol(),Period(),time3,10,temp);
      //--- 在第二点左侧9柱设置第一点
      time1=temp[0];
     }
//--- 如果第一点的价格没有设置，则它与第三点的价格相等
   if(!price1)
      price1=price3;
//--- 如果第二点的时间没有设置，它则位于第三点左侧的7个柱
   if(!time2)
      time2=temp[2];
//--- 如果第二个点的价格没有设置，则低于第一个点移动250点
   if(!price2)
      price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变对象定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制斐波纳契扩展线的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建斐波纳契扩展线
   if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy/10;
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 移动点
      if(!FiboExpansionPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy/2;
//--- 移动第三个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p3>1)
         p3-=1;
      //--- 移动点
      if(!FiboExpansionPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy*4/5;
//--- 移动第二个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p2<accuracy-1)
         p2+=1;
      //--- 移动点
      if(!FiboExpansionPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 删除图表对象
   FiboExpansionDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ELLIOTWAVE5

艾略特波浪动力

注意

对于"艾略特波浪动力"，可以启用/禁用通过线连接点的模式（OBJPROP_DRAWLINES 属性），以及设置波浪定位的水平（从 ENUM_ELLIOT_WAVE_DEGREE 枚举）。

示例

下面的脚本创建和移动图表上的艾略特波浪动力。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Elliott Motive Wave\"."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string                InpName="ElliotWave5";   // 对象名称
input int                   InpDate1=10;             // 第1个点的日期，%
input int                   InpPrice1=90;            // 第1个点的价格，%
input int                   InpDate2=20;             // 第2个点的日期，%
input int                   InpPrice2=40;            // 第2个点的价格，%
input int                   InpDate3=30;             // 第3个点的日期，%
input int                   InpPrice3=60;            // 第3个点的价格，%
input int                   InpDate4=40;             // 第4个点的日期，%
```

© 2000-2025, MetaQuotes Ltd.

```
input int                      InpPrice4=10;              // 第4个点的价格，%
input int                      InpDate5=60;               // 第5个点的日期，%
input int                      InpPrice5=40;              // 第5个点的价格，%
input ENUM_ELLIOT_WAVE_DEGREE  InpDegree=ELLIOTT_MINOR;   // 水平
input bool                     InpDrawLines=true;         // 显示线
input color                    InpColor=clrRed;           // 线的颜色
input ENUM_LINE_STYLE          InpStyle=STYLE_DASH;       // 线的风格
input int                      InpWidth=2;                // 线的宽度
input bool                     InpBack=false;             // 背景对象
input bool                     InpSelection=true;         // 突出移动
input bool                     InpHidden=true;            // 隐藏在对象列表
input long                     InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建“艾略特波浪动力”                                |
//+------------------------------------------------------------------+
bool ElliotWave5Create(const long                  chart_ID=0,              // 图表
                       const string                name="ElliotWave5",      // 波浪名
                       const int                   sub_window=0,            // 子窗口
                       datetime                    time1=0,                 // 第一个
                       double                      price1=0,                // 第一个
                       datetime                    time2=0,                 // 第二个
                       double                      price2=0,                // 第二个
                       datetime                    time3=0,                 // 第三个
                       double                      price3=0,                // 第三个
                       datetime                    time4=0,                 // 第四个
                       double                      price4=0,                // 第四个
                       datetime                    time5=0,                 // 第五个
                       double                      price5=0,                // 第五个
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // 度数
                       const bool                  draw_lines=true,         // 显示线
                       const color                 clr=clrRed,              // 对象颜
                       const ENUM_LINE_STYLE       style=STYLE_SOLID,       // 线的风
                       const int                   width=1,                 // 线的宽
                       const bool                  back=false,              // 在背景
                       const bool                  selection=true,          // 突出移
                       const bool                  hidden=true,             // 隐藏在
                       const long                  z_order=0)               // 鼠标单
  {
//--- 若未设置则设置定位点的坐标
   ChangeElliotWave5EmptyPoints(time1,price1,time2,price2,time3,price3,time4,price4,ti
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建“艾略特波浪动力”
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE5,sub_window,time1,price1,time2,price2
      price3,time4,price4,time5,price5))
     {
      Print(__FUNCTION__,
            ": failed to create \"Elliott Motive Wave\"! Error code = ",GetLastError()
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 设置程度 波动大小）
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- 启用 （true） 或禁用 （false） 展示线型的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- 设置对象的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//---设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 （false） 或背景 （true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 （true） 或禁用 （false） 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏（true） 或显示 （false） 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动艾略特波浪动力定位点                                                  |
//+------------------------------------------------------------------+
bool ElliotWave5PointChange(const long   chart_ID=0,          // 图表 ID
                            const string name="ElliotWave5", // 对象名称
                            const int    point_index=0,      // 定位点指数
                            datetime     time=0,             // 定位点时间坐标
                            double       price=0)            // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除艾略特波浪动力                                                 |
//+------------------------------------------------------------------+
bool ElliotWave5Delete(const long   chart_ID=0,         // 图表 ID
                       const string name="ElliotWave5") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Elliott Motive Wave\"! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查艾略特波浪动力定位点的值和                                     |
//| 为空点设置默认的值                                                 |
//+------------------------------------------------------------------+
void ChangeElliotWave5EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3,
                                  datetime &time4,double &price4,
                                  datetime &time5,double &price5)
  {
//--- 接收最近10柱开盘时间的数组
   datetime temp[];
   ArrayResize(temp,10);
//--- 接收数据
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- 接收当前图表一点的值
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第一点的时间没有设置，它则位于最后柱左侧的9个柱
   if(!time1)
      time1=temp[0];
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于最后柱左侧的7个柱
   if(!time2)
      time2=temp[2];
//--- 如果第二个点的价格没有设置，则低于第一个点移动300点
   if(!price2)
```

© 2000-2025, MetaQuotes Ltd.

```
      price2=price1-300*point;
//--- 如果第三点的时间没有设置，它则位于最后柱左侧的5个柱
   if(!time3)
      time3=temp[4];
//--- 如果第三个点的价格没有设置，则低于第一个点移动250点
   if(!price3)
      price3=price1-250*point;
//--- 如果第四点的时间没有设置，它则位于最后柱左侧的3个柱
   if(!time4)
      time4=temp[6];
//--- 如果第四个点的价格没有设置，则低于第一个点移动550点
   if(!price4)
      price4=price1-550*point;
//--- 如果第四点的时间没有设置，它将位于最后的柱
   if(!time5)
      time5=temp[9];
//--- 如果第五个点的价格没有设置，则低于第一个点移动450点
   if(!price5)
      price5=price1-450*point;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100 ||
      InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
      InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变对象定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Failed to copy time values! Error code = ",GetLastError());
       return;
      }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制艾略特波浪动力的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int d4=InpDate4*(bars-1)/100;
   int d5=InpDate5*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
   int p4=InpPrice4*(accuracy-1)/100;
   int p5=InpPrice5*(accuracy-1)/100;
//--- 创建艾略特波浪动力
   if(!ElliotWave5Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pr
      date[d4],price[p4],date[d5],price[p5],InpDegree,InpDrawLines,InpColor,InpStyle,I
      InpBack,InpSelection,InpHidden,InpZOrder))
      {
       return;
      }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 移动第五个定位点
   for(int i=0;i<v_steps;i++)
      {
       //--- 使用下面的值
       if(p5<accuracy-1)
          p5+=1;
       //--- 移动点
       if(!ElliotWave5PointChange(0,InpName,4,date[d5],price[p5]))
          return;
       //--- 检查脚本操作是否已经强制禁用
       if(IsStopped())
          return;
       //--- 重画图表
```

© 2000-2025, MetaQuotes Ltd.

```
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy/5;
//--- 移动第二个和第三个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p2<accuracy-1)
         p2+=1;
      if(p3>1)
         p3-=1;
      //--- 切换点
      if(!ElliotWave5PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      if(!ElliotWave5PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy*4/5;
//--- 移动第一个和第四个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      if(p4<accuracy-1)
         p4+=1;
      //--- 切换点
      if(!ElliotWave5PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave5PointChange(0,InpName,3,date[d4],price[p4]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
```

© 2000-2025, MetaQuotes Ltd.

```
   Sleep(1000);
//--- 删除图表对象
   ElliotWave5Delete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ELLIOTWAVE3

埃利奥特纠正波

注意

对于"埃利奥特纠正波"，可以启用/禁用通过线连接点的模式（OBJPROP_DRAWLINES 属性），以及设置波浪定位的水平（从 ENUM_ELLIOT_WAVE_DEGREE 枚举）。

示例

下面的脚本创建和移动图表上的埃利奥特纠正波。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Elliott Correction Wave\" graphical object."
#property description "Anchor point coordinates are set in percentage of the chart's w
#property description "size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string                  InpName="ElliotWave3";    // 对象名称
input int                     InpDate1=10;              // 第1个点的日期，%
input int                     InpPrice1=90;             // 第1个点的价格，%
input int                     InpDate2=30;              // 第2个点的日期，%
input int                     InpPrice2=10;             // 第2个点的价格，%
input int                     InpDate3=50;              // 第3个点的日期，%
input int                     InpPrice3=40;             // 第3个点的价格，%
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // 水平
```

© 2000-2025, MetaQuotes Ltd.

```
input bool                    InpDrawLines=true;       // 显示线
input color                   InpColor=clrRed;         // 线的颜色
input ENUM_LINE_STYLE         InpStyle=STYLE_DASH;     // 线的风格
input int                     InpWidth=2;              // 线的宽度
input bool                    InpBack=false;           // 背景对象
input bool                    InpSelection=true;       // 突出移动
input bool                    InpHidden=true;          // 隐藏在对象列表
input long                    InpZOrder=0;             // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建“埃利奥特纠正波”                                |
//+------------------------------------------------------------------+
bool ElliotWave3Create(const long                  chart_ID=0,               // 图表I
                       const string                name="ElliotWave3",       // 波浪名
                       const int                   sub_window=0,             // 子窗口
                       datetime                    time1=0,                  // 第一个
                       double                      price1=0,                 // 第一个
                       datetime                    time2=0,                  // 第二个
                       double                      price2=0,                 // 第二个
                       datetime                    time3=0,                  // 第三个
                       double                      price3=0,                 // 第三个
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // 度数
                       const bool                  draw_lines=true,          // 显示线
                       const color                 clr=clrRed,               // 对象颜
                       const ENUM_LINE_STYLE       style=STYLE_SOLID,        // 线的风
                       const int                   width=1,                  // 线的宽
                       const bool                  back=false,               // 在背景
                       const bool                  selection=true,           // 突出移
                       const bool                  hidden=true,              // 隐藏在
                       const long                  z_order=0)                // 鼠标单
  {
//--- 若未设置则设置定位点的坐标
   ChangeElliotWave3EmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建“埃利奥特纠正波”
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE3,sub_window,time1,price1,time2,price2
     {
      Print(__FUNCTION__,
            ": failed to create \"Elliott Correction Wave\"! Error code = ",GetLastErr
      return(false);
     }
//--- 设置程度（波动大小）
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- 启用（true）或禁用（false）展示线型的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- 设置对象的颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//---设置线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出通道移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动埃利奥特纠正波定位点                                         |
//+------------------------------------------------------------------+
bool ElliotWave3PointChange(const long   chart_ID=0,          // 图表 ID
                            const string name="ElliotWave3", // 对象名称
                            const int    point_index=0,      // 定位点指数
                            datetime     time=0,             // 定位点时间坐标
                            double       price=0)            // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除埃利奥特纠正波                                               |
//+------------------------------------------------------------------+
bool ElliotWave3Delete(const long   chart_ID=0,         // 图表ID
                       const string name="ElliotWave3") // 对象名称
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Elliott Correction Wave\"! Error code = ",GetLastErr
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查埃利奥特纠正波定位点的值和                                   |
//| 为空点设置默认的值                                               |
//+------------------------------------------------------------------+
void ChangeElliotWave3EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3)
  {
//--- 接收最近10柱开盘时间的数组
   datetime temp[];
   ArrayResize(temp,10);
//--- 接收数据
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- 接收当前图表一点的值
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第一点的时间没有设置，它则位于最后柱左侧的9个柱
   if(!time1)
      time1=temp[0];
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于最后柱左侧的5个柱
   if(!time2)
      time2=temp[4];
//--- 如果第二个点的价格没有设置，则低于第一个点移动300点
   if(!price2)
      price2=price1-300*point;
//--- 如果第三点的时间没有设置，它则位于最后柱左侧的1个柱
   if(!time3)
      time3=temp[8];
//---如果第三个点的价格没有设置，则低于第一个点移动200点
   if(!price3)
      price3=price1-200*point;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变对象定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制埃利奥特纠正波的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建埃利奥特纠正波
   if(!ElliotWave3Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
      InpDegree,InpDrawLines,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidde
     {
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 移动第三个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p3<accuracy-1)
         p3+=1;
      //--- 移动点
      if(!ElliotWave3PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy*4/5;
//--- 移动第一个和第二个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      if(p2<accuracy-1)
         p2+=1;
      //--- 切换点
      if(!ElliotWave3PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave3PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表对象
   ElliotWave3Delete(0,InpName);
   ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_RECTANGLE

矩形。

注意

对于矩形，颜色填充模式可以使用OBJPROP_FILL属性来设置。

示例

下面的脚本创建和移动图表上的矩形。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates rectangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Rectangle"; // 矩形名称
input int              InpDate1=40;         // 第1个点的日期，%
input int              InpPrice1=40;        // 第1个点的价格，%
input int              InpDate2=60;         // 第2个点的日期，%
input int              InpPrice2=60;        // 第2个点的价格，%
input color            InpColor=clrRed;     // 矩形颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH; // 矩形线的风格
input int              InpWidth=2;          // 矩形线的宽度
input bool             InpFill=true;        // 填充矩形颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input bool             InpBack=false;       // 背景矩形
input bool             InpSelection=true;   // 突出移动
input bool             InpHidden=true;      // 隐藏在对象列表
input long             InpZOrder=0;         // 鼠标单击优先
//+------------------------------------------------------------------+
//| 通过已给的坐标创建矩形                                           |
//+------------------------------------------------------------------+
bool RectangleCreate(const long             chart_ID=0,         // 图表 ID
                     const string           name="Rectangle",   // 矩形名称
                     const int              sub_window=0,       // 子窗口指数
                     datetime               time1=0,            // 第一个点的时间
                     double                 price1=0,           // 第一个点的价格
                     datetime               time2=0,            // 第二个点的时间
                     double                 price2=0,           // 第二个点的价格
                     const color            clr=clrRed,         // 矩形颜色
                     const ENUM_LINE_STYLE  style=STYLE_SOLID,  // 矩形线的风格
                     const int              width=1,            // 矩形线的宽度
                     const bool             fill=false,         // 填充矩形颜色
                     const bool             back=false,         // 在背景中
                     const bool             selection=true,     // 突出移动
                     const bool             hidden=true,        // 隐藏在对象列表
                     const long             z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建矩形
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle! Error code = ",GetLastError());
      return(false);
     }
//--- 设置矩形颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置矩形线的风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置矩形线的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 启用（true）或禁用（false）填充矩形的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）突出矩形移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动矩形定位点                                                   |
//+------------------------------------------------------------------+
bool RectanglePointChange(const long   chart_ID=0,        // 图表 ID
                          const string name="Rectangle", // 矩形名称
                          const int    point_index=0,    // 定位点指数
                          datetime     time=0,           // 定位点时间坐标
                          double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除矩形                                                         |
//+------------------------------------------------------------------+
bool RectangleDelete(const long   chart_ID=0,        // 图表 ID
                     const string name="Rectangle") // 矩形名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除矩形
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete rectangle! Error code = ",GetLastError());
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查矩形定位点的值和为空点设置                                   |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二个点的价格没有设置，则低于第一个点移动300点
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变矩形定位点的坐标
   datetime date[];
   double   price[];
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制矩形的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- 创建矩形
   if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动矩形的定位点
//--- 循环计数器
   int h_steps=bars/2;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- 切换点
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
```

© 2000-2025, MetaQuotes Ltd.

```
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- 切换点
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表矩形
   RectangleDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_TRIANGLE

三角形。

注意

对于三角形，颜色填充模式可以使用 OBJPROP_FILL 属性来设置。

示例

下面的脚本创建和移动图表上的三角形。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates triangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Triangle";         // 三角形名称
input int              InpDate1=25;                // 第1个点的日期，%
input int              InpPrice1=50;               // 第一个点的价格，%
input int              InpDate2=70;                // 第2个点的日期， %
input int              InpPrice2=70;               // 第2个点的价格，%
input int              InpDate3=65;                // 第3个点的日期，%
input int              InpPrice3=20;               // 第3个点的价格，%
input color            InpColor=clrRed;            // 三角形颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 三角形线条风格
```

© 2000-2025, MetaQuotes Ltd.

```
input int              InpWidth=2;                // 三角形线条宽度
input bool             InpFill=false;             // 三角形填充颜色
input bool             InpBack=false;             // 背景三角形
input bool             InpSelection=true;         // 突出移动
input bool             InpHidden=true;            // 隐藏在对象列表
input long             InpZOrder=0;               // 鼠标单击优先
//+------------------------------------------------------------------+
//|通过已给的坐标创建三角形                                          |
//+------------------------------------------------------------------+
bool TriangleCreate(const long            chart_ID=0,        //图表 ID
                    const string          name="Triangle",   //三角形名称
                    const int             sub_window=0,      // 子窗口指数
                    datetime              time1=0,           // 第一个点的时间
                    double                price1=0,          // 第一个点的价格
                    datetime              time2=0,           // 第二个点的时间
                    double                price2=0,          // 第二个点的价格
                    datetime              time3=0,           //第三个点的时间
                    double                price3=0,          // 第三个点的价格
                    const color           clr=clrRed,        //三角形颜色
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // 三角形线条风格
                    const int             width=1,           // 三角形线条宽度
                    const bool            fill=false,        // 三角形填充颜色
                    const bool            back=false,        // 在背景中
                    const bool            selection=true,    // 突出移动
                    const bool            hidden=true,       // 隐藏在对象列表
                    const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeTriangleEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 通过已给的坐标创建三角形
   if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create a triangle! Error code = ",GetLastError());
      return(false);
     }
//--- 设置三角形颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置三角形线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置三角形线条宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 启用 (true) 或禁用 (false) 填充三角形的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出三角形移动的模式
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动三角形定位点                                                 |
//+------------------------------------------------------------------+
bool TrianglePointChange(const long   chart_ID=0,       // 图表 ID
                         const string name="Triangle", // 三角形名称
                         const int    point_index=0,   // 定位点指数
                         datetime     time=0,          // 定位点时间坐标
                         double       price=0)         // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除三角形                                                       |
//+------------------------------------------------------------------+
bool TriangleDelete(const long   chart_ID=0,       // 图表 ID
                    const string name="Triangle") // 三角形名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除三角形
   if(!ObjectDelete(chart_ID,name))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__,
            ": failed to delete the ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查三角形定位点的值和为空点设置                                 |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeTriangleEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2,
                               datetime &time3,double &price3)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二个点的价格没有设置，则低于第一个点移动300点
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第三个点的时间没有设置，则它与第二个点的日期相一致
   if(!time3)
      time3=time2;
//--- 如果第三点的价格没有设置，则它与第一点的价格相等
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Error! Incorrect values of input parameters!");
       return;
      }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变三角形定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制三角形的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建三角形
   if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动三角形定位点
//--- 循环计数器
   int v_steps=accuracy*3/10;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 移动第一个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1>1)
         p1-=1;
      //--- 移动点
      if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars*9/20-1;
//--- 移动第二个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d2>1)
         d2-=1;
      //--- 移动点
      if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   v_steps=accuracy/4;
//--- 移动第三个定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p3<accuracy-1)
         p3+=1;
      //--- 移动点
      if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))
         return;
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除三角形
   TriangleDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ELLIPSE

椭圆形。

注意

对于椭圆形，颜色填充模式可以使用 OBJPROP_FILL 属性来设置。

示例

下面的脚本创建和移动图表上的椭圆形。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates ellipse on the chart."
#property description "Anchor point coordinates are set"
#property description "in percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Ellipse";          // 椭圆形名称
input int              InpDate1=30;                // 第1个点的日期，%
input int              InpPrice1=20;               // 第1个点的价格，%
input int              InpDate2=70;                // 第2个点的日期， %
input int              InpPrice2=80;               // 第2个点的价格，%
input int              InpDate3=50;                // 第3个点的日期，%
input int              InpPrice3=60;               // 第3个点的价格，%
input color            InpColor=clrRed;            // 椭圆形颜色
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // 椭圆形线条风格
input int              InpWidth=2;                 // 椭圆形线条的宽度
```

© 2000-2025, MetaQuotes Ltd.

```
input bool            InpFill=false;              // 椭圆形填充颜色
input bool            InpBack=false;              // 背景椭圆形
input bool            InpSelection=true;          // 突出移动
input bool            InpHidden=true;             // 隐藏在对象列表
input long            InpZOrder=0;                // 鼠标单击优先
//+------------------------------------------------------------------+
//| 由给定坐标创建椭圆形                                               |
//+------------------------------------------------------------------+
bool EllipseCreate(const long            chart_ID=0,        // 图表ID
                   const string          name="Ellipse",    // 椭圆形名称
                   const int             sub_window=0,      // 子窗口指数
                   datetime              time1=0,           // 第一个点的时间
                   double                price1=0,          // 第一个点的价格
                   datetime              time2=0,           // 第二个点的时间
                   double                price2=0,          // 第二个点的价格
                   datetime              time3=0,           // 第三个点的时间
                   double                price3=0,          // 第三个点的价格
                   const color           clr=clrRed,        // 椭圆形颜色
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // 椭圆形线条风格
                   const int             width=1,           // 椭圆形线条的宽度
                   const bool            fill=false,        // 椭圆形填充颜色
                   const bool            back=false,        // 在背景中
                   const bool            selection=true,    // 突出移动
                   const bool            hidden=true,       // 隐藏在对象列表
                   const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeEllipseEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- 重置错误的值
   ResetLastError();
//--- 由给定坐标创建椭圆形
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time2,price2,tir
     {
      Print(__FUNCTION__,
            ": failed to create an ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- 设置椭圆形颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置椭圆形线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置椭圆形线条的宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 启用 (true) 或禁用 (false) 填充椭圆形的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 突出椭圆形移动的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动椭圆形定位点                                                 |
//+------------------------------------------------------------------+
bool EllipsePointChange(const long   chart_ID=0,     // 图表ID
                        const string name="Ellipse", // 椭圆形名称
                        const int    point_index=0,  // 定位点指数
                        datetime     time=0,         // 定位点时间坐标
                        double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除椭圆形                                                       |
//+------------------------------------------------------------------+
bool EllipseDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="Ellipse") // 椭圆形名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除椭圆形
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
```

© 2000-2025, MetaQuotes Ltd.

```
            ": failed to delete an ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查椭圆形定位点的值和为空点设置                                 |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2,
                              datetime &time3,double &price3)
  {
//--- 如果第一点的时间没有设置，它将位于当前柱
   if(!time1)
      time1=TimeCurrent();
//--- 如果第一点的价格没有设置，则它将用卖价值
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 如果第二点的时间没有设置，它则位于第二点左侧的9个柱
   if(!time2)
     {
      //--- 接收最近10柱开盘时间的数组
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- 在第一点左侧9柱设置第二点
      time2=temp[0];
     }
//--- 如果第二个点的价格没有设置，则低于第一个点移动300点
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- 如果第三个点的时间没有设置，则它与第二个点的日期相一致
   if(!time3)
      time3=time2;
//--- 如果第三点的价格没有设置，则它与第一点的价格相等
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//---设置和改变椭圆形定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制椭圆形的点
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- 创建椭圆形
   if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动椭圆形定位点
//--- 循环计数器
   int v_steps=accuracy/5;
//--- 移动第一个和第二个定位点
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- 切换点
      if(!EllipsePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!EllipsePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int h_steps=bars/5;
//--- 移动第三个定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d3>1)
         d3-=1;
      //--- 移动点
      if(!EllipsePointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除椭圆形
   EllipseDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_THUMB_UP

拇指向上符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的拇指向上符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Thumbs Up\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="ThumbUp";     // 符号名称
input int                InpDate=75;            // 定位点日期在%
input int                InpPrice=25;           // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_TOP;  // 定位类型
input color              InpColor=clrRed;       // 符号颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;    // 边界线条风格
input int                InpWidth=5;            // 符号大小
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpBack=false;         // 背景符号
input bool               InpSelection=true;     // 突出移动
input bool               InpHidden=true;        // 隐藏对象列表中
input long               InpZOrder=0;           // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建拇指向上符号                                                 |
//+------------------------------------------------------------------+
bool ArrowThumbUpCreate(const long              chart_ID=0,           // 图表 ID
                        const string            name="ThumbUp",       // 符号名称
                        const int               sub_window=0,         // 子窗口指数
                        datetime                time=0,               // 定位点时间
                        double                  price=0,              // 定位点价格
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                        const color             clr=clrRed,           // 符号颜色
                        const ENUM_LINE_STYLE   style=STYLE_SOLID,    // 边界线条风格
                        const int               width=3,              // 符号大小
                        const bool              back=false,           // 在背景中
                        const bool              selection=true,       // 突出移动
                        const bool              hidden=true,          // 隐藏在对象列表
                        const long              z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false） 图形对象名称
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowThumbUpMove(const long   chart_ID=0,      // 图表 ID
                      const string name="ThumbUp", // 对象名称
                      datetime     time=0,         // 定位点时间坐标
                      double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变拇指向上符号定位类型                                         |
//+------------------------------------------------------------------+
bool ArrowThumbUpAnchorChange(const long              chart_ID=0,        // 图表 ID
                              const string            name="ThumbUp",    // 对象名称
                              const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 删除拇指向上符号                                                 |
//+------------------------------------------------------------------+
bool ArrowThumbUpDelete(const long   chart_ID=0,     // 图表 ID
                        const string name="ThumbUp") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表创建拇指向上符号
   if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int h_steps=bars/4;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d>1)
         d-=1;
      //--- 移动点
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int v_steps=accuracy/4;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p<accuracy-1)
         p+=1;
      //--- 移动点
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 改变定位点相对于符号的位置
   ArrowThumbUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowThumbUpDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_THUMB_DOWN

拇指向下符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的拇指向下符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Thumbs Down\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="ThumbDown";      // 符号名称
input int                InpDate=25;               // 定位点日期在%
input int                InpPrice=75;              // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_BOTTOM;  // 定位类型
input color              InpColor=clrRed;          // 符号颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;       // 边界线条风格
```

© 2000-2025, MetaQuotes Ltd.

```
input int                InpWidth=5;               // 符号大小
input bool               InpBack=false;            // 背景符号
input bool               InpSelection=true;        // 突出移动
input bool               InpHidden=true;           // 隐藏对象列表中
input long               InpZOrder=0;              // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建拇指向下符号                                                 |
//+------------------------------------------------------------------+
bool ArrowThumbDownCreate(const long              chart_ID=0,            // 图表 ID
                          const string            name="ThumbDown",      // 符号名称
                          const int               sub_window=0,          // 子窗口指数
                          datetime                time=0,                // 定位点时间
                          double                  price=0,               // 定位点价格
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                          const color             clr=clrRed,            // 符号颜色
                          const ENUM_LINE_STYLE   style=STYLE_SOLID,     // 边界线条风格
                          const int               width=3,               // 符号大小
                          const bool              back=false,            // 背景中
                          const bool              selection=true,        // 突出移动
                          const bool              hidden=true,           // 隐藏在对象列
                          const long              z_order=0)             // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowThumbDownMove(const long   chart_ID=0,        // 图表 ID
                        const string name="ThumbDown", // 对象名称
                        datetime     time=0,           // 定位点时间坐标
                        double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变手指向下符号定位类型                                         |
//+------------------------------------------------------------------+
bool ArrowThumbDownAnchorChange(const long              chart_ID=0,        // 图表 ID
                                const string            name="ThumbDown",  // 对象名称
                                const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除拇指向下符号                                                 |
//+------------------------------------------------------------------+
bool ArrowThumbDownDelete(const long   chart_ID=0,       // 图表 ID
                          const string name="ThumbDown") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置和改变符号定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建拇指向下符号
   if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int h_steps=bars/4;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
```

© 2000-2025, MetaQuotes Ltd.

```
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 循环计数器
   int v_steps=accuracy/4;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p>1)
         p-=1;
      //--- 移动点
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 改变定位点相对于符号的位置
   ArrowThumbDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowThumbDownDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_UP

箭头向上符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的箭头向上符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Arrow Up\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string              InpName="ArrowUp";     // 符号名称
input int                 InpDate=25;            // 定位点日期在%
input int                 InpPrice=25;           // 定位点价格在 %
input ENUM_ARROW_ANCHOR   InpAnchor=ANCHOR_TOP;  // 定位类型
input color               InpColor=clrRed;       // 符号颜色
input ENUM_LINE_STYLE     InpStyle=STYLE_DOT;    // 边界线条风格
input int                 InpWidth=5;            // 符号大小
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpBack=false;         // 背景符号
input bool               InpSelection=false;    // 突出移动
input bool               InpHidden=true;        // 隐藏对象列表中
input long               InpZOrder=0;           // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建箭头向上符号                                                 |
//+------------------------------------------------------------------+
bool ArrowUpCreate(const long              chart_ID=0,           // 图表 ID
                   const string            name="ArrowUp",       // 符号名称
                   const int               sub_window=0,         // 子窗口指数
                   datetime                time=0,               // 定位点时间
                   double                  price=0,              // 定位点价格
                   const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                   const color             clr=clrRed,           // 符号颜色
                   const ENUM_LINE_STYLE   style=STYLE_SOLID,    // 边界线条风格
                   const int               width=3,              // 符号大小
                   const bool              back=false,           // 在背景中
                   const bool              selection=true,       // 突出移动
                   const bool              hidden=true,          // 隐藏在对象列表
                   const long              z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false） 图形对象名称
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowUpMove(const long   chart_ID=0,     // 图表 ID
                 const string name="ArrowUp", // 对象名称
                 datetime     time=0,         // 定位点时间坐标
                 double       price=0)        // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变箭头向下符号定位类型                                         |
//+------------------------------------------------------------------+
bool ArrowUpAnchorChange(const long              chart_ID=0,        // 图表 ID
                         const string            name="ArrowUp",    // 对象名称
                         const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位点位置
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 删除箭头向上符号                                                 |
//+------------------------------------------------------------------+
bool ArrowUpDelete(const long   chart_ID=0,     // 图表 ID
                   const string name="ArrowUp") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表创建箭头向上符号
   if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p<accuracy-1)
         p+=1;
      //--- 移动点
      if(!ArrowUpMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 1 秒延迟
   Sleep(1000);
//--- 改变定位点相对于符号的位置
   ArrowUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowUpDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_DOWN

箭头向下符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的箭头向下符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Arrow Down\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="ArrowDown";      // 符号名称
input int                InpDate=75;               // 定位点日期在 %
input int                InpPrice=75;              // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_BOTTOM;  // 定位类型
input color              InpColor=clrRed;          // 符号颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;       // 边界线条风格
input int                InpWidth=5;               // 符号大小
input bool               InpBack=false;            // 背景符号
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpSelection=false;       // 突出移动
input bool               InpHidden=true;           // 隐藏对象列表中
input long               InpZOrder=0;              // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建箭头向下符号                                                 |
//+------------------------------------------------------------------+
bool ArrowDownCreate(const long              chart_ID=0,            // 图表 ID
                     const string            name="ArrowDown",      // 符号名称
                     const int               sub_window=0,          // 子窗口指数
                     datetime                time=0,                // 定位点时间
                     double                  price=0,               // 定位点价格
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                     const color             clr=clrRed,            // 符号颜色
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,     // 边界线条风格
                     const int               width=3,               // 符号大小
                     const bool              back=false,            // 在背景中
                     const bool              selection=true,        // 突出移动
                     const bool              hidden=true,           // 隐藏在对象列表
                     const long              z_order=0)             // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowDownMove(const long   chart_ID=0,        // 图表 ID
                   const string name="ArrowDown", // 对象名称
                   datetime     time=0,           // 定位点时间坐标
                   double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变箭头向下符号定位类型                                         |
//+------------------------------------------------------------------+
bool ArrowDownAnchorChange(const long              chart_ID=0,        // 图表 ID
                           const string            name="ArrowDown",  // 对象名称
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位点位置
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 删除箭头向下符号                                                 |
//+------------------------------------------------------------------+
bool ArrowDownDelete(const long   chart_ID=0,       // 图表 ID
                     const string name="ArrowDown") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
   datetime date[];
```

© 2000-2025, MetaQuotes Ltd.

```
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表创建箭头向下符号
   if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int v_steps=accuracy/2;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p>1)
         p-=1;
      //--- 移动点
      if(!ArrowDownMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 1 秒延迟
   Sleep(1000);
//--- 改变定位点相对于符号的位置
   ArrowDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- 重画图表
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowDownDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_STOP

停止符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的停止符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Stop\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="ArrowStop";      // 符号名称
input int                InpDate=10;               // 定位点日期在 %
input int                InpPrice=50;              // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_BOTTOM;  // 定位类型
input color              InpColor=clrRed;          // 符号颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;       // 边界线条风格
input int                InpWidth=5;               // 符号大小
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpBack=false;           // 背景符号
input bool               InpSelection=false;      // 突出移动
input bool               InpHidden=true;          // 隐藏对象列表中
input long               InpZOrder=0;             // 鼠标单击优先
//+------------------------------------------------------------------+
//| Create Stop sign                                                 |
//+------------------------------------------------------------------+
bool ArrowStopCreate(const long              chart_ID=0,           // 图表 ID
                     const string            name="ArrowStop",     // 符号名称
                     const int               sub_window=0,         // 子窗口指数
                     datetime                time=0,               // 定位点时间
                     double                  price=0,              // 定位点价格
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                     const color             clr=clrRed,           // 符号颜色
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // 边界线条风格
                     const int               width=3,              // 符号大小
                     const bool              back=false,           // 在背景中
                     const bool              selection=true,       // 突出移动
                     const bool              hidden=true,          // 隐藏在对象列表
                     const long              z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Stop\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false） 图形对象名称
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowStopMove(const long   chart_ID=0,        // 图表 ID
                   const string name="ArrowStop", // 对象名称
                   datetime     time=0,           // 定位点时间坐标
                   double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变停止符号定位类型                                             |
//+------------------------------------------------------------------+
bool ArrowStopAnchorChange(const long                chart_ID=0,         // 图表 ID
                           const string              name="ArrowStop",   // 对象名称
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位点位置
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 删除停止符号                                                     |
//+------------------------------------------------------------------+
bool ArrowStopDelete(const long   chart_ID=0,        // 图表 ID
                     const string name="ArrowStop") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Stop\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建停止符号
   if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int h_steps=bars*2/5;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
      // 0.025 秒延迟
      Sleep(25);
     }
//--- 改变定位点相对于符号的位置
   ArrowStopAnchorChange(0,InpName,ANCHOR_TOP);
//--- 重画图表
   ChartRedraw();
//--- 循环计数器
   h_steps=bars*2/5;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.025 秒延迟
      Sleep(25);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowStopDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_CHECK

查验符号。

注意

相对于符号的定位点位置可以从ENUM_ARROW_ANCHOR枚举选择。

在MetaEditor编写代码时，大型符号（大于5）只能通过设置适当的OBJPROP_WIDTH属性值来创建。

示例

下面的脚本创建和移动图表上的查验符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Check\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="ArrowCheck"; // 符号名称
input int                InpDate=10;           // 定位点日期在 %
input int                InpPrice=50;          // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_TOP; // 定位类型
input color              InpColor=clrRed;      // 符号颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;   // 边界线条风格
input int                InpWidth=5;           // 符号大小
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpBack=false;         // 背景符号
input bool               InpSelection=false;    // 突出移动
input bool               InpHidden=true;        // 隐藏对象列表中
input long               InpZOrder=0;           // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建检查符号                                                     |
//+------------------------------------------------------------------+
bool ArrowCheckCreate(const long              chart_ID=0,           // 图表标ID
                      const string            name="ArrowCheck",    // 符号名称
                      const int               sub_window=0,         // 子窗口指数
                      datetime                time=0,               // 定位点时间
                      double                  price=0,              // 定位点价格
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位类型
                      const color             clr=clrRed,           // 符号颜色
                      const ENUM_LINE_STYLE   style=STYLE_SOLID,    // 边界线条风格
                      const int               width=3,              // 符号大小
                      const bool              back=false,           // 在背景中
                      const bool              selection=true,       // 突出移动
                      const bool              hidden=true,          // 隐藏在对象列表
                      const long              z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Check\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置符号大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动符号的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false） 图形对象名称
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowCheckMove(const long   chart_ID=0,         // 图表 ID
                    const string name="ArrowCheck", // 对象名称
                    datetime     time=0,            // 定位点时间坐标
                    double       price=0)           // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 更改检查定位类型                                                 |
//+------------------------------------------------------------------+
bool ArrowCheckAnchorChange(const long                chart_ID=0,         // 图表 ID
                            const string              name="ArrowCheck", // 对象名称
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 删除检查符号                                                     |
//+------------------------------------------------------------------+
bool ArrowCheckDelete(const long   chart_ID=0,        // 图表 ID
                      const string name="ArrowCheck") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Check\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制符号的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建检查符号
   if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点和改变其相对于符号的位置
//--- 循环计数器
   int h_steps=bars*2/5;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
      // 0.025 秒延迟
      Sleep(25);
     }
//--- 改变定位点相对于符号的位置
   ArrowCheckAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- 重画图表
   ChartRedraw();
//--- 循环计数器
   h_steps=bars*2/5;
//--- 移动定位点
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.025 秒延迟
      Sleep(25);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表符号
   ArrowCheckDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_LEFT_PRICE

向左价格标签

示例

下面的脚本创建和移动图表上的向左价格标签。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates the left price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string            InpName="LeftPrice";  // 价格标签名称
input int               InpDate=100;          // 定位点日期在%
input int               InpPrice=10;          // 定位点价格在 %
input color             InpColor=clrRed;      // 价格标签颜色
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // 边界线条风格
input int               InpWidth=2;           // 价格标签大小
input bool              InpBack=false;        // 背景标签
input bool              InpSelection=true;    // 突出移动
input bool              InpHidden=true;       // 隐藏对象列表中
input long              InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建向左价格标签                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool ArrowLeftPriceCreate(const long            chart_ID=0,        // 图表 ID
                          const string          name="LeftPrice",  // 价格标签名称
                          const int             sub_window=0,      // 子窗口指数
                          datetime              time=0,            // 定位点时间
                          double                price=0,           // 定位点价格
                          const color           clr=clrRed,        // 价格标签颜色
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // 边界线条风格
                          const int             width=1,           // 价格标签大小
                          const bool            back=false,        // 在背景中
                          const bool            selection=true,    // 突出移动
                          const bool            hidden=true,       // 隐藏在对象列表
                          const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建价格标签
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create the left price label! Error code = ",GetLastError());
      return(false);
     }
//--- 设置标签颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置标签大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowLeftPriceMove(const long   chart_ID=0,       // 图表 ID
```

© 2000-2025, MetaQuotes Ltd.

```
                        const string name="LeftPrice", // 标签名称
                        datetime     time=0,           // 定位点时间坐标
                        double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 从图表删除向左价格标签                                           |
//+------------------------------------------------------------------+
bool ArrowLeftPriceDelete(const long   chart_ID=0,       // 图表 ID
                          const string name="LeftPrice") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the left price label! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 用于设置和改变标签定位点坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制标签的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建向左价格标签
   if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy*4/5;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p<accuracy-1)
         p+=1;
      //--- 移动点
      if(!ArrowLeftPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除标签
   ArrowLeftPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_RIGHT_PRICE

向右价格标签。

示例

下面的脚本创建和移动图表上的向右价格标签。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates the right price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string              InpName="RightPrice"; // 价格标签名称
input int                 InpDate=0;            // 定位点日期在%
input int                 InpPrice=90;          // 定位点价格在 %
input color               InpColor=clrRed;      // 价格标签颜色
input ENUM_LINE_STYLE     InpStyle=STYLE_SOLID; // 边界线条风格
input int                 InpWidth=2;           // 价格标签大小
input bool                InpBack=false;        // 背景标签
input bool                InpSelection=true;    // 突出移动
input bool                InpHidden=true;       // 隐藏对象列表中
input long                InpZOrder=0;          // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建向右价格标签                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool ArrowRightPriceCreate(const long             chart_ID=0,        // 图表 ID
                           const string           name="RightPrice", // 价格标签名称
                           const int              sub_window=0,      // 子窗口指数
                           datetime               time=0,            // 定位点时间
                           double                 price=0,           // 定位点价格
                           const color            clr=clrRed,        // 价格标签颜色
                           const ENUM_LINE_STYLE  style=STYLE_SOLID, // 边界线条风格
                           const int              width=1,           // 价格标签大小
                           const bool             back=false,        // 在背景中
                           const bool             selection=true,    // 突出移动
                           const bool             hidden=true,       // 隐藏在对象列表
                           const long             z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建价格标签
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create the right price label! Error code = ",GetLastError());
      return(false);
     }
//--- 设置标签颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置标签大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowRightPriceMove(const long   chart_ID=0,        // 图表 ID
```

© 2000-2025, MetaQuotes Ltd.

```
                           const string name="RightPrice", // 标签名称
                           datetime     time=0,            // 定位点时间坐标
                           double       price=0)           // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 从图表删除向右价格标签                                           |
//+------------------------------------------------------------------+
bool ArrowRightPriceDelete(const long   chart_ID=0,        // 图表 ID
                           const string name="RightPrice") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the right price label! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 用于设置和改变标签定位点坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制标签的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建向右价格标签
   if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
```

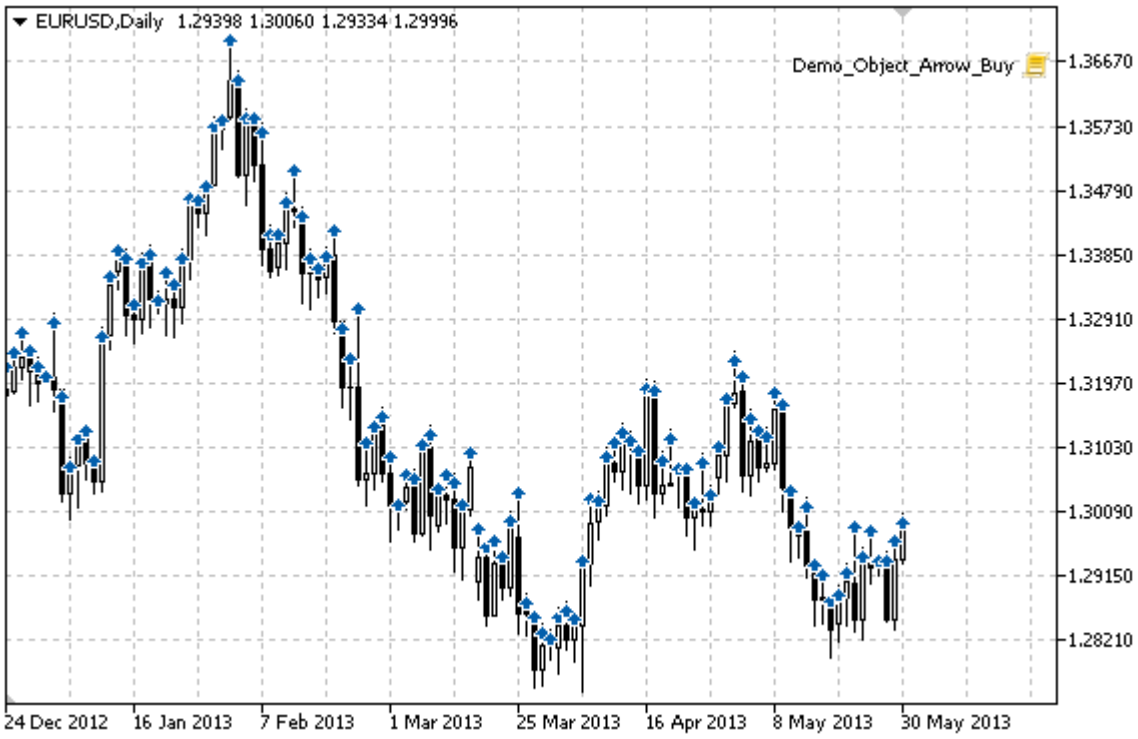
© 2000-2025, MetaQuotes Ltd.

```
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动定位点
//--- 循环计数器
   int v_steps=accuracy*4/5;
//--- 移动定位点
   for(int i=0;i<v_steps;i++)
     {
      //--- 使用下面的值
      if(p>1)
         p-=1;
      //--- 移动点
      if(!ArrowRightPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除标签
   ArrowRightPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_BUY

买入符号。




示例

下面的脚本创建和移动图表上的买入符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Buy\" signs in the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input color InpColor=C'3,95,172'; // 颜色符号
//+------------------------------------------------------------------+
//| 创建买入符号                                                     |
//+------------------------------------------------------------------+
bool ArrowBuyCreate(const long            chart_ID=0,        // 图表 ID
                    const string          name="ArrowBuy",   // 符号名称
                    const int             sub_window=0,      // 子窗口指数
                    datetime              time=0,            // 定位点时间
                    double                price=0,           // 定位点价格
                    const color           clr=C'3,95,172',   // 符号颜色
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // 线条风格 (当高亮显示时)
                    const int             width=1,           // 线条大小 (当高亮显示时)
                    const bool            back=false,        // 在背景中
                    const bool            selection=false,   // 突出移动
                    const bool            hidden=true,       // 隐藏在对象列表
```

© 2000-2025, MetaQuotes Ltd.

```
                    const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Buy\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线条风格 (当高亮显示时)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线条大小 (当高亮显示时)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动符号的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowBuyMove(const long   chart_ID=0,       // 图表 ID
                  const string name="ArrowBuy", // 对象名称
                  datetime     time=0,          // 定位点时间坐标
                  double       price=0)         // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除买入符号                                                     |
//+------------------------------------------------------------------+
bool ArrowBuyDelete(const long   chart_ID=0,       // 图表ID
                    const string name="ArrowBuy") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Buy\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // 存储可见柱日期的数组
   double   low[];  // 存储可见柱最低价格的数组
   double   high[]; // 存储可见柱最高价格的数组
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 内存分配
```

© 2000-2025, MetaQuotes Ltd.

```
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写最低价格的数组
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- 填写最高价格的数组
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError());
      return;
     }
//--- 为每个可见柱创建最低点的买入符号
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 为每个可见柱移动买入符号到最高点
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 删除买入符号
```

© 2000-2025, MetaQuotes Ltd.

```
    for(int i=0;i<bars;i++)
      {
       if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
          return;
       //--- 重画图表
       ChartRedraw();
       // 0.05 秒延迟
       Sleep(50);
      }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW_SELL

卖出符号。

示例

下面的脚本创建和移动图表上的卖出符号。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Sell\" signs in the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input color InpColor=C'225,68,29'; // 颜色符号
//+------------------------------------------------------------------+
//| 创建卖出符号                                                     |
//+------------------------------------------------------------------+
bool ArrowSellCreate(const long            chart_ID=0,        // 图表 ID
                     const string          name="ArrowSell",  // 符号名称
                     const int             sub_window=0,      // 子窗口指数
                     datetime              time=0,            // 定位点时间
                     double                price=0,           // 定位点价格
                     const color           clr=C'225,68,29',  // 符号颜色
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // 线条风格（当高亮显示时）
                     const int             width=1,           // 线条大小（当高亮显示时）
                     const bool            back=false,        // 在背景中
                     const bool            selection=false,   // 突出移动
                     const bool            hidden=true,       // 隐藏在对象列表
```

© 2000-2025, MetaQuotes Ltd.

```
                      const long            z_order=0)         // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建符号
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Sell\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 设置符号颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置线条风格 (高亮显示时)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置线条大小 (高亮显示时)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动符号的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowSellMove(const long   chart_ID=0,        // 图表 ID
                   const string name="ArrowSell", // 对象名称
                   datetime     time=0,           // 定位点时间坐标
                   double       price=0)          // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除卖出符号                                                     |
//+------------------------------------------------------------------+
bool ArrowSellDelete(const long   chart_ID=0,       // 图表 ID
                     const string name="ArrowSell") // 符号名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除符号
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Sell\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // 存储可见柱日期的数组
   double   low[];  // 存储可见柱最低价格的数组
   double   high[]; // 存储可见柱最高价格的数组
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 内存分配
```

© 2000-2025, MetaQuotes Ltd.

```
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写最低价格的数组
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- 填写最高价格的数组
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError());
      return;
     }
//--- 为每个可见柱创建最高点的卖出符号
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpColor))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 为每个可见柱移动卖出符号到最低点
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 删除卖出符号
```

© 2000-2025, MetaQuotes Ltd.

```
    for(int i=0;i<bars;i++)
      {
       if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
          return;
       //--- 重画图表
       ChartRedraw();
       // 0.05 秒延迟
       Sleep(50);
      }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_ARROW

箭头对象。

注意

相对于对象的定位点位置可以从 ENUM_ARROW_ANCHOR来选择。

在MetaEditor编写代码时，大型箭头（大于5）只能通过设置适当的 OBJPROP_WIDTH 属性值来创建。

需要的箭头类型可以通过设置一种 Wingdings 字体的符号代码来选择。

示例

下面的脚本创建图表上的箭头对象和改变其类型。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates a random arrow in the chart window."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="Arrow";         // 箭头名称
input int                InpDate=50;              // 定位点日期在 %
input int                InpPrice=50;             // 定位点价格在 %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_TOP;    // 定位类型
input color              InpColor=clrDodgerBlue; // 箭头颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;   // 边界线条风格
input int               InpWidth=10;            // 箭头大小
input bool              InpBack=false;          // 背景箭头
input bool              InpSelection=false;     // 突出移动
input bool              InpHidden=true;         // 隐藏对象列表中
input long              InpZOrder=0;            // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建箭头                                                         |
//+------------------------------------------------------------------+
bool ArrowCreate(const long              chart_ID=0,           // 图表 ID
                 const string            name="Arrow",         // 箭头名称
                 const int               sub_window=0,         // 子窗口指数
                 datetime                time=0,               // 定位点时间
                 double                  price=0,              // 定位点价格
                 const uchar             arrow_code=252,       // 箭头代码
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // 定位点位置
                 const color             clr=clrRed,           // 箭头颜色
                 const ENUM_LINE_STYLE   style=STYLE_SOLID,    // 边界线条风格
                 const int               width=3,              // 箭头大小
                 const bool              back=false,           // 在背景中
                 const bool              selection=true,       // 突出移动
                 const bool              hidden=true,          // 隐藏在对象列表
                 const long              z_order=0)            // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeArrowEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建箭头
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- 设置箭头代码
   ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置箭头颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置箭头大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动箭头的模式
//--- 当使用ObjectCreate函数创建图形对象时，对象不能
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 默认下突出并移动。在这个方法中，默认选择参数
//--- true 可以突出移动对象
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool ArrowMove(const long   chart_ID=0,   // 图表 ID
               const string name="Arrow", // 对象名称
               datetime     time=0,       // 定位点时间坐标
               double       price=0)      // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变箭头代码                                                     |
//+------------------------------------------------------------------+
bool ArrowCodeChange(const long   chart_ID=0,   // 图表 ID
                     const string name="Arrow", // 对象名称
                     const uchar  code=252)     // 箭头代码
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变箭头代码
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))
     {
      Print(__FUNCTION__,
```

© 2000-2025, MetaQuotes Ltd.

```
            ": failed to change the arrow code! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变定向类型                                                     |
//+------------------------------------------------------------------+
bool ArrowAnchorChange(const long              chart_ID=0,        // 图表 ID
                       const string            name="Arrow",      // 对象名称
                       const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // 定位类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除箭头                                                         |
//+------------------------------------------------------------------+
bool ArrowDelete(const long   chart_ID=0,   // 图表 ID
                 const string name="Arrow") // 箭头名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除箭头
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 价格数组大小
   int accuracy=1000;
//--- 存储要使用的日期和价格值的数组
//--- 设置和改变符号定位点的坐标
   datetime date[];
   double   price[];
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写价格数组
//--- 找出图表的最高值和最低值
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- 定义变化的价格并填写该数组
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- 定义绘制箭头的点
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- 在图表上创建箭头
   if(!ArrowCreate(0,InpName,0,date[d],price[p],32,InpAnchor,InpColor,
```

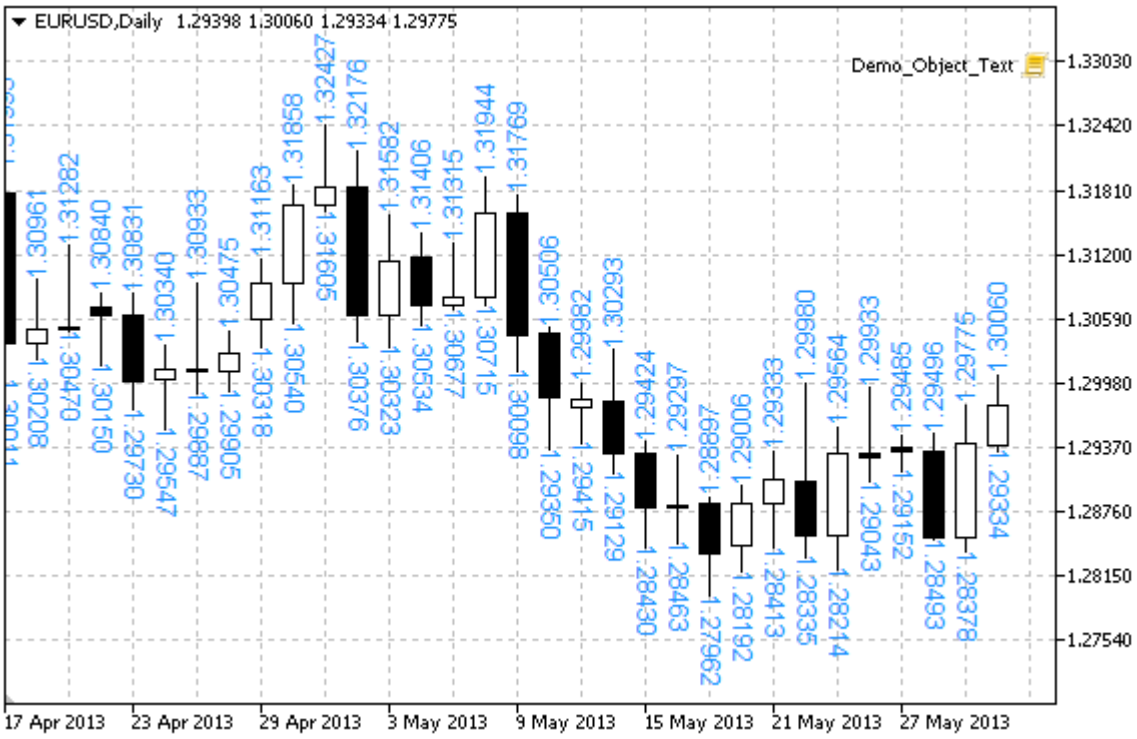
© 2000-2025, MetaQuotes Ltd.

```
       InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
      {
       return;
      }
//--- 重画图表
   ChartRedraw();
//--- 考虑循环中创建箭头的所有情况
   for(int i=33;i<256;i++)
     {
      if(!ArrowCodeChange(0,InpName,(uchar)i))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 延迟半秒
      Sleep(500);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 从图表删除箭头
   ArrowDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_TEXT

文本对象。




注意

相对于文本的定位点位置可以从 ENUM_ANCHOR_POINT 枚举选择。您也可以使用 OBJPROP_ANGLE 属性来改变文本定位角。

示例

下面的脚本创建图表上的多个文本对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Text\" graphical object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string              InpFont="Arial";          // 字体
input int                 InpFontSize=10;           // 字体大小
input color               InpColor=clrRed;          // 颜色
input double              InpAngle=90.0;            // 倾斜角度
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_LEFT;    // 定位类型
input bool                InpBack=false;            // 背景对象
input bool                InpSelection=false;       // 突出移动
input bool                InpHidden=true;           // 隐藏对象列表中
input long                InpZOrder=0;              // 鼠标单击优先
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 创建文本对象                                                     |
//+------------------------------------------------------------------+
bool TextCreate(const long              chart_ID=0,                 // 图表 ID
                const string            name="Text",                // 对象名称
                const int               sub_window=0,               // 子窗口指数
                datetime                time=0,                     // 定位点时间
                double                  price=0,                    // 定位点价格
                const string            text="Text",                // 文本本身
                const string            font="Arial",               // 字体
                const int               font_size=10,               // 字体大小
                const color             clr=clrRed,                 // 颜色
                const double            angle=0.0,                  // 文本倾斜
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER,   // 定位类型
                const bool              back=false,                 // 在背景中
                const bool              selection=false,            // 突出移动
                const bool              hidden=true,                // 隐藏在对象列表
                const long              z_order=0)                  // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeTextEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建文本对象
   if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Text\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 设置文本
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- 设置文本字体
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- 设置字体大小
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- 设置文本的倾斜角
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动对象的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动定位点                                                       |
//+------------------------------------------------------------------+
bool TextMove(const long   chart_ID=0,  // 图表 ID
              const string name="Text", // 对象名称
              datetime     time=0,      // 定位点时间坐标
              double       price=0)     // 定位点价格坐标
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变对象文本                                                     |
//+------------------------------------------------------------------+
bool TextChange(const long   chart_ID=0,  // 图表 ID
                const string name="Text", // 对象名称
                const string text="Text") // 文本
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象文本
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 删除文本对象                                                     |
//+------------------------------------------------------------------+
bool TextDelete(const long   chart_ID=0,  // 图表 ID
                const string name="Text") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Text\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeTextEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // 存储可见柱日期的数组
   double   low[];  // 存储可见柱最低价格的数组
   double   high[]; // 存储可见柱最高价格的数组
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
//--- 填写最低价格的数组
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- 填写最高价格的数组
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError());
      return;
     }
//--- 定义文本显示的频率
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- 定义步骤
   int step=1;
   switch(scale)
     {
      case 0:
         step=12;
         break;
      case 1:
         step=6;
         break;
      case 2:
         step=4;
         break;
      case 3:
         step=2;
         break;
     }
//--- 创建最高和最低柱的值的文本 间隔)。
   for(int i=0;i<bars;i+=step)
     {
      //--- 创建文本
      if(!TextCreate(0,"TextHigh_"+(string)i,0,date[i],high[i],DoubleToString(high[i],
         InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
        {
         return;
        }
      if(!TextCreate(0,"TextLow_"+(string)i,0,date[i],low[i],DoubleToString(low[i],5),
         InpColor,-InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
        {
         return;
        }
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
```

© 2000-2025, MetaQuotes Ltd.

```
            return;
         //--- 重画图表
         ChartRedraw();
         // 0.05 秒延迟
         Sleep(50);
        }
//--- 半秒延迟
   Sleep(500);
//--- 删除文本
   for(int i=0;i<bars;i+=step)
     {
      if(!TextDelete(0,"TextHigh_"+(string)i))
         return;
      if(!TextDelete(0,"TextLow_"+(string)i))
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_LABEL

标签对象。

注意

相对于标签的定位点位置可以从 ENUM_ANCHOR_POINT 枚举选择。定位点坐标用像素设置。

您也可以从 ENUM_BASE_CORNER 枚举选择文本标签定位角。

示例

下面的脚本创建和移动图表上的编辑对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Label\" graphical object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string            InpName="Label";          // 标签名称
input int               InpX=150;                 // X-轴距离
input int               InpY=150;                 // Y-轴距离
input string            InpFont="Arial";          // 字体
input int               InpFontSize=14;           // 字体大小
input color             InpColor=clrRed;          // 颜色
input double            InpAngle=0.0;             // 倾斜角度
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // 定位类型
input bool              InpBack=false;            // 背景对象
input bool              InpSelection=true;        // 突出移动
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpHidden=true;          // 隐藏对象列表中
input long              InpZOrder=0;             // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建文本标签                                                     |
//+------------------------------------------------------------------+
bool LabelCreate(const long              chart_ID=0,               // 图表 ID
                 const string            name="Label",             // 标签名称
                 const int               sub_window=0,             // 子窗口指数
                 const int               x=0,                      // X 坐标
                 const int               y=0,                      // Y 坐标
                 const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // 图表定位角
                 const string            text="Label",             // 文本
                 const string            font="Arial",             // 字体
                 const int               font_size=10,             // 字体大小
                 const color             clr=clrRed,               // 颜色
                 const double            angle=0.0,                // 文本倾斜
                 const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // 定位类型
                 const bool              back=false,               // 在背景中
                 const bool              selection=false,          // 突出移动
                 const bool              hidden=true,              // 隐藏在对象列表
                 const long              z_order=0)                // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建文本标签
   if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create text label! Error code = ",GetLastError());
      return(false);
     }
//--- 设置标签坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置相对于定义点坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置文本
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- 设置文本字体
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- 设置字体大小
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- 设置文本的倾斜角
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 显示前景 (false) 或背景 (true)
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动文本标签                                                     |
//+------------------------------------------------------------------+
bool LabelMove(const long   chart_ID=0,   // 图表 ID
               const string name="Label", // 标签名称
               const int    x=0,          // X 坐标
               const int    y=0)          // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动文本标签
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变绑定标签的图表角落                                           |
//+------------------------------------------------------------------+
bool LabelChangeCorner(const long             chart_ID=0,               // 图表 ID
                       const string           name="Label",             // 标签名称
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // 图表定位角
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位角
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
```

```
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变标签文本                                                     |
//+------------------------------------------------------------------+
bool LabelTextChange(const long   chart_ID=0,   // 图表 ID
                     const string name="Label", // 对象名称
                     const string text="Text")  // 文本
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象文本
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除文本标签                                                     |
//+------------------------------------------------------------------+
bool LabelDelete(const long   chart_ID=0,   // 图表 ID
                 const string name="Label") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a text label! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在本地变量存储标签坐标
   int x=InpX;
   int y=InpY;
//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 检查输入参数的正确性
   if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 准备标签的初始文本
   string text;
   StringConcatenate(text,"Upper left corner: ",x,",",y);
//--- 在图表上创建文本标签
   if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,InpFontSize,
      InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待0.5秒
   ChartRedraw();
   Sleep(500);
//--- 同时移动标签和改变其文本
//--- 轴的迭代次数
   int h_steps=(int)(x_distance/2-InpX);
   int v_steps=(int)(y_distance/2-InpY);
//--- 向下移动标签
   for(int i=0;i<v_steps;i++)
     {
      //--- 改变坐标
      y+=2;
      //--- 移动标签并更改其文本
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- 半秒延迟
   Sleep(500);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 向右移动标签
   for(int i=0;i<h_steps;i++)
     {
      //--- 改变坐标
      x+=2;
      //--- 移动标签并更改其文本
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- 半秒延迟
   Sleep(500);
//--- 向上移动标签
   for(int i=0;i<v_steps;i++)
     {
      //--- 改变坐标
      y-=2;
      //--- 移动标签并更改其文本
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- 半秒延迟
   Sleep(500);
//--- 向左移动标签
   for(int i=0;i<h_steps;i++)
     {
      //--- 改变坐标
      x-=2;
      //--- 移动标签并更改其文本
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- 半秒延迟
   Sleep(500);
//--- 现在，通过改变定位角来移动点的位置
//--- 移动到左下角
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
      return;
//--- 改变标签文本
   StringConcatenate(text,"Lower left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- 重绘图表并等待2秒
   ChartRedraw();
   Sleep(2000);
//--- 移动到右下角
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
      return;
//--- 改变标签文本
   StringConcatenate(text,"Lower right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- 重绘图表并等待2秒
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartRedraw();
   Sleep(2000);
//--- 移动到右上角
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_UPPER))
      return;
//--- 改变标签文本
   StringConcatenate(text,"Upper right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- 重绘图表并等待2秒
   ChartRedraw();
   Sleep(2000);
//--- 移动到左上角
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_UPPER))
      return;
//--- 改变标签文本
   StringConcatenate(text,"Upper left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- 重绘图表并等待2秒
   ChartRedraw();
   Sleep(2000);
//--- 删除标签
   LabelDelete(0,InpName);
//--- 重画图表并等待0.5秒
   ChartRedraw();
   Sleep(500);
//---
  }
//+------------------------------------------------------------------+
//| 函数移动对象并改变其文本                                         |
//+------------------------------------------------------------------+
bool MoveAndTextChange(const int x,const int y,string text)
  {
//--- 移动标签
   if(!LabelMove(0,InpName,x,y))
      return(false);
//--- 改变标签文本
   StringConcatenate(text,text,x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return(false);
//--- 检查脚本操作是否已经强制禁用
   if(IsStopped())
      return(false);
//--- 重画图表
   ChartRedraw();
// 0.01 秒延迟
   Sleep(10);
//--- 推出函数
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_BUTTON

按钮对象。

注意

定位点坐标用像素设置。您可以从 ENUM_BASE_CORNER选择按钮定位角。

示例

下面的脚本创建和移动图表上的按钮对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates the button on the chart."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string            InpName="Button";           // 按钮名称
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER; // 图表定位角
input string            InpFont="Arial";            // 字体
input int               InpFontSize=14;             // 字体大小
input color             InpColor=clrBlack;          // 文本颜色
input color             InpBackColor=C'236,233,216'; // 背景色
input color             InpBorderColor=clrNONE;     // 边界颜色
input bool              InpState=false;             // 出版/发布
input bool              InpBack=false;              // 背景对象
input bool              InpSelection=false;         // 突出移动
input bool              InpHidden=true;             // 隐藏在对象列表
input long              InpZOrder=0;                // 鼠标单击优先
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 创建按钮                                                         |
//+------------------------------------------------------------------+
bool ButtonCreate(const long              chart_ID=0,               // 图表 ID
                  const string            name="Button",            // 按钮名称
                  const int               sub_window=0,             // 子窗口指数
                  const int               x=0,                      // X 坐标
                  const int               y=0,                      // Y 坐标
                  const int               width=50,                 // 按钮宽度
                  const int               height=18,                // 按钮高度
                  const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // 图表定位角
                  const string            text="Button",            // 文本
                  const string            font="Arial",             // 字体
                  const int               font_size=10,             // 字体大小
                  const color             clr=clrBlack,             // 文本颜色
                  const color             back_clr=C'236,233,216',  // 背景色
                  const color             border_clr=clrNONE,       // 边界颜色
                  const bool              state=false,              // 出版/发布
                  const bool              back=false,               // 在背景中
                  const bool              selection=false,          // 突出移动
                  const bool              hidden=true,              // 隐藏在对象列表
                  const long              z_order=0)                // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建按钮
   if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create the button! Error code = ",GetLastError());
      return(false);
     }
//--- 设置按钮坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置按钮大小
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置相对于定义点坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置文本
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- 设置文本字体
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- 设置字体大小
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- 设置文本颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置背景颜色
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- 设置边界颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动按钮的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动按钮                                                         |
//+------------------------------------------------------------------+
bool ButtonMove(const long   chart_ID=0,    // 图表 ID
                const string name="Button", // 按钮名称
                const int    x=0,           // X 坐标
                const int    y=0)           // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动按钮
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the button! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the button! Error code = ",GetLastError
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变按钮大小                                                     |
//+------------------------------------------------------------------+
bool ButtonChangeSize(const long   chart_ID=0,    // 图表 ID
                      const string name="Button", // 按钮名称
                      const int    width=50,      // 按钮宽度
                      const int    height=18)     // 按钮高度
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置错误的值
   ResetLastError();
//--- 改变按钮大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the button width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the button height! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变绑定按钮的图表角落                                           |
//+------------------------------------------------------------------+
bool ButtonChangeCorner(const long             chart_ID=0,               // 图表 ID
                        const string           name="Button",            // 按钮名称
                        const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // 图表定位角
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变定位角
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变按钮文本                                                     |
//+------------------------------------------------------------------+
bool ButtonTextChange(const long   chart_ID=0,    // 图表 ID
                      const string name="Button", // 按钮名称
                      const string text="Text")   // 文本
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象文本
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
```

```
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除按钮                                                         |
//+------------------------------------------------------------------+
bool ButtonDelete(const long   chart_ID=0,    // 图表 ID
                  const string name="Button") // 按钮名称
  {
//--- 重置错误的值
   ResetLastError();
//---删除按钮
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the button! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 定义改变按钮大小的步骤
   int x_step=(int)x_distance/32;
   int y_step=(int)y_distance/32;
//--- 设置按钮坐标及其大小
   int x=(int)x_distance/32;
```

© 2000-2025, MetaQuotes Ltd.

```
   int y=(int)y_distance/32;
   int x_size=(int)x_distance*15/16;
   int y_size=(int)y_distance*15/16;
//--- 创建按钮
   if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFont,InpFontSiz
      InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,InpHidden,Inp
     {
      return;
     }
//--- 重画图表
   ChartRedraw();
//--- 减少循环中的按钮
   int i=0;
   while(i<13)
     {
      //--- 延迟半秒
      Sleep(500);
      //--- 转换按钮到出版状态
      ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
      //--- 重绘图表，等候0.2 秒
      ChartRedraw();
      Sleep(200);
      //--- 重新定义坐标和按钮大小
      x+=x_step;
      y+=y_step;
      x_size-=x_step*2;
      y_size-=y_step*2;
      //--- 减少按钮
      ButtonMove(0,InpName,x,y);
      ButtonChangeSize(0,InpName,x_size,y_size);
      //--- 按钮返回发布状态
      ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
      //--- 重画图表
      ChartRedraw();
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 增加循环计数器
      i++;
     }
//--- 半秒延迟
   Sleep(500);
//---删除按钮
   ButtonDelete(0,InpName);
   ChartRedraw();
//--- 等待1秒
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_CHART

图表对象。

注意

在可视化测试期间不支持（不显示）"OBJ_CHART"类型对象。

定位点坐标用像素设置。您可以从 ENUM_BASE_CORNER 枚举选择定位角。

可以为图表对象选择交易品种，周期和比例尺。也可以启用/禁用价格比例和日期展示模式。

示例

下面的脚本创建和移动图表上的图表对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Chart\" object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string            InpName="Chart";              // 对象名称
input string            InpSymbol="EURUSD";           // 交易品种
input ENUM_TIMEFRAMES   InpPeriod=PERIOD_H1;          // 周期
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER; // 定位角
input int               InpScale=2;                   // 比例
input bool              InpDateScale=true;            // 时间比例显示
input bool              InpPriceScale=true;           // 价格比例显示
input color             InpColor=clrRed;              // 高亮显示时的边界颜色
```

© 2000-2025, MetaQuotes Ltd.

```
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT;    // 高亮显示时的线条风格
input int              InpPointWidth=1;              // 移动点大小
input bool             InpBack=false;                // 背景对象
input bool             InpSelection=true;            // 突出移动
input bool             InpHidden=true;               // 隐藏在对象列表
input long             InpZOrder=0;                  // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建图表对象                                                     |
//+------------------------------------------------------------------+
bool ObjectChartCreate(const long              chart_ID=0,               // 图表 ID
                       const string            name="Chart",             // 对象名称
                       const int               sub_window=0,             // 子窗口指数
                       const string            symbol="EURUSD",          // 交易品种
                       const ENUM_TIMEFRAMES   period=PERIOD_H1,         // 周期
                       const int               x=0,                      // X 坐标
                       const int               y=0,                      // Y 坐标
                       const int               width=300,                // 宽度
                       const int               height=200,               // 高度
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // 定位角
                       const int               scale=2,                  // 比例
                       const bool              date_scale=true,          // 时间比例显示
                       const bool              price_scale=true,         // 价格比例显示
                       const color             clr=clrRed,               // 高亮显示时的
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // 高亮显示时的
                       const int               point_width=1,            // 移动点大小
                       const bool              back=false,               // 在背景中
                       const bool              selection=false,          // 突出移动
                       const bool              hidden=true,              // 隐藏在对象列
                       const long              z_order=0)                // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建图表对象
   if(!ObjectCreate(chart_ID,name,OBJ_CHART,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Chart\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 设置对象坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置对象大小
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置相对于定义点坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置交易品种
   ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置周期
   ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period);
//--- 设置比例
   ObjectSetInteger(chart_ID,name,OBJPROP_CHART_SCALE,scale);
//--- 显示 (true) 或隐藏 (false) 时间比例
   ObjectSetInteger(chart_ID,name,OBJPROP_DATE_SCALE,date_scale);
//--- 显示 (true) 或隐藏 (false) 价格比例
   ObjectSetInteger(chart_ID,name,OBJPROP_PRICE_SCALE,price_scale);
//--- 设置对象高亮模式启动时的边界颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置对象高亮模式启动时的边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置移动对象的定位点大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 设置交易品种和图表对象时间帧                                     |
//+------------------------------------------------------------------+
bool ObjectChartSetSymbolAndPeriod(const long            chart_ID=0,        // 图表 ID
                                   const string          name="Chart",      // 对象名称
                                   const string          symbol="EURUSD",   // 交易品种
                                   const ENUM_TIMEFRAMES period=PERIOD_H1) // 时间帧
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置图表对象交易品种和时间帧
   if(!ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol))
     {
      Print(__FUNCTION__,
            ": failed to set a symbol for \"Chart\" object! Error code = ",GetLastErro
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period))
     {
      Print(__FUNCTION__,
            ": failed to set a period for \"Chart\" object! Error code = ",GetLastErro
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动图表对象                                                     |
//+------------------------------------------------------------------+
bool ObjectChartMove(const long   chart_ID=0,   // 图表 ID（非图表对象）
                     const string name="Chart", // 对象名称
                     const int    x=0,          // X 坐标
                     const int    y=0)          // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动对象
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变图表对象大小                                                 |
//+------------------------------------------------------------------+
bool ObjectChartChangeSize(const long   chart_ID=0,   // 图表 ID（非图表对象）
                           const string name="Chart", // 对象名称
                           const int    width=300,    // 宽度
                           const int    height=200)   // 高度
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the width of \"Chart\" object! Error code = ",GetLastE
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
```

© 2000-2025, MetaQuotes Ltd.

```
            ": failed to change the height of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 返回图表对象 ID                                                  |
//+------------------------------------------------------------------+
long ObjectChartGetID(const long   chart_ID=0,   // 图表 ID（非图表对象）
                      const string name="Chart") // 对象名称
  {
//--- 准备变量获得图表对象 ID
   long id=-1;
//--- 重置错误的值
   ResetLastError();
//--- 获得ID
   if(!ObjectGetInteger(chart_ID,name,OBJPROP_CHART_ID,0,id))
     {
      Print(__FUNCTION__,
            ": failed to get \"Chart\" object's ID! Error code = ",GetLastError());
     }
//--- 返回结果
   return(id);
  }
//+------------------------------------------------------------------+
//| 删除图表对象                                                     |
//+------------------------------------------------------------------+
bool ObjectChartDelete(const long   chart_ID=0,   // 图表 ID（非图表对象）
                       const string name="Chart") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//---删除按钮
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Chart\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获得市场报价交易品种的数量
```

© 2000-2025, MetaQuotes Ltd.

```
   int  symbols=SymbolsTotal(true);
//--- 检查交易品种列表中是否存在指定名称的交易品种
   bool exist=false;
   for(int i=0;i<symbols;i++)
      if(InpSymbol==SymbolName(i,true))
        {
         exist=true;
         break;
        }
   if(!exist)
     {
      Print("Error! ",InpSymbol," symbol is not present in \"Market Watch\"!");
      return;
     }
//--- 检查导入参数的正确性
   if(InpScale<0 || InpScale>5)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }

//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 设置图表对象坐标及其大小
   int x=(int)x_distance/16;
   int y=(int)y_distance/16;
   int x_size=(int)x_distance*7/16;
   int y_size=(int)y_distance*7/16;
//--- 创建图表对象
   if(!ObjectChartCreate(0,InpName,0,InpSymbol,InpPeriod,x,y,x_size,y_size,InpCorner,I
      InpPriceScale,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
```

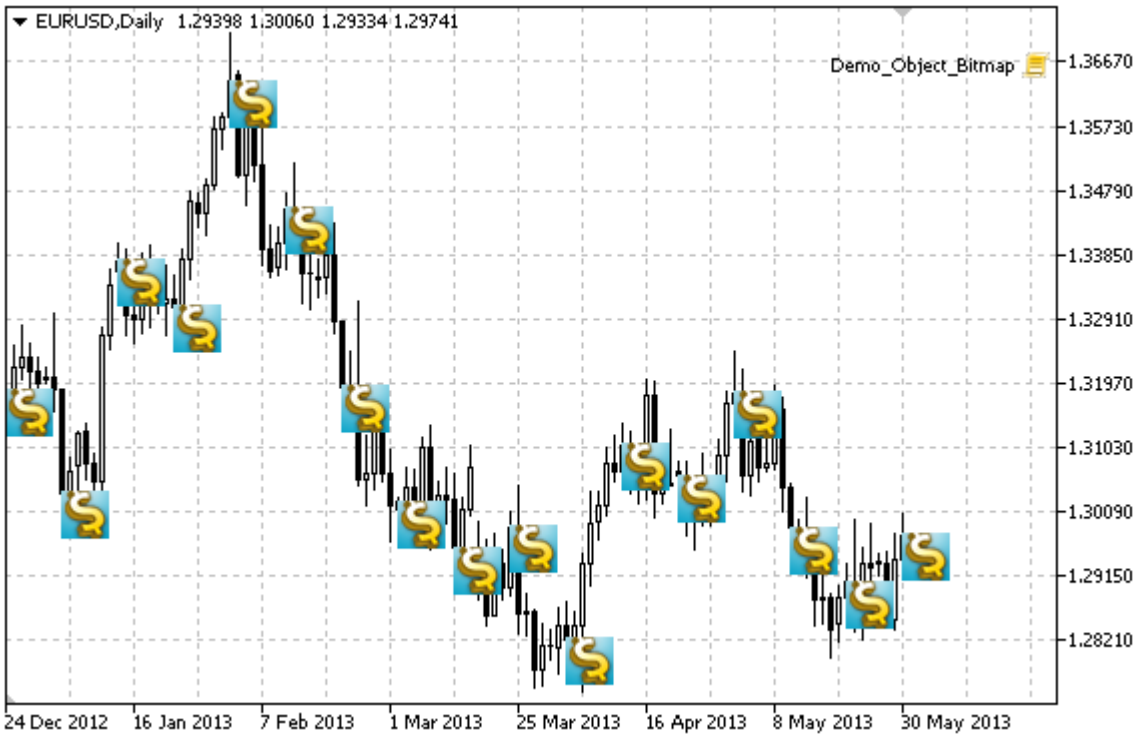
© 2000-2025, MetaQuotes Ltd.

```
//--- 拉伸图表对象
   int steps=(int)MathMin(x_distance*7/16,y_distance*7/16);
   for(int i=0;i<steps;i++)
     {
      //--- resize
      x_size+=1;
      y_size+=1;
      if(!ObjectChartChangeSize(0,InpName,x_size,y_size))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重绘图表并等待0.01秒
      ChartRedraw();
      Sleep(10);
     }
//--- 半秒延迟
   Sleep(500);
//--- 改变图表时间帧
   if(!ObjectChartSetSymbolAndPeriod(0,InpName,InpSymbol,PERIOD_M1))
      return;
   ChartRedraw();
//--- 3秒延迟
   Sleep(3000);
//--- 删除对象
   ObjectChartDelete(0,InpName);
   ChartRedraw();
//--- 等待1秒
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_BITMAP

位图对象。




注意

对于位图对象，您可以选择图像的可视范围。

示例

下面的脚本创建图表上的多个位图。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates a bitmap in the chart window."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpFile="\\Images\\dollar.bmp"; // 位图文件名称
input int              InpWidth=24;                    // 可见范围 X 坐标
input int              InpHeight=24;                   // 可见范围 Y 坐标
input int              InpXOffset=4;                   // 可见范围沿着 X 轴移动
input int              InpYOffset=4;                   // 可见范围沿着 Y 轴移动
input color            InpColor=clrRed;                // 高亮显示时的边界颜色
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;           // 高亮显示时的线条风格
input int              InpPointWidth=1;                // 移动点大小
input bool             InpBack=false;                  // 背景对象
input bool             InpSelection=false;             // 突出移动
input bool             InpHidden=true;                 // 隐藏在对象列表
```

© 2000-2025, MetaQuotes Ltd.

```
input long            InpZOrder=0;                     // 鼠标单击优先
//+------------------------------------------------------------------+
//| 在图表窗口创建位图                                                 |
//+------------------------------------------------------------------+
bool BitmapCreate(const long            chart_ID=0,         // 图表 ID
                  const string          name="Bitmap",      // 位图名称
                  const int             sub_window=0,       // 子窗口指数
                  datetime              time=0,             // 定位点时间
                  double                price=0,            // 定位点价格
                  const string          file="",            // 位图文件名称
                  const int             width=10,           // 可见范围 X 坐标
                  const int             height=10,          // 可见范围 Y 坐标
                  const int             x_offset=0,         // 可见范围沿着 X 轴移动
                  const int             y_offset=0,         // 可见范围沿着 Y 轴移动
                  const color           clr=clrRed,         // 高亮显示时的边界颜色
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // 高亮显示时的线条风格
                  const int             point_width=1,      // 移动点大小
                  const bool            back=false,         // 在背景中
                  const bool            selection=false,    // 突出移动
                  const bool            hidden=true,        // 隐藏在对象列表
                  const long            z_order=0)          // 鼠标单击优先
  {
//--- 若未设置则设置定位点的坐标
   ChangeBitmapEmptyPoint(time,price);
//--- 重置错误的值
   ResetLastError();
//--- 创建位图
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create a bitmap in the chart window! Error code = ",GetLastEr
      return(false);
     }
//--- 设置图像文件路径
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to download the image! Error code = ",GetLastError());
      return(false);
     }
//--- 设置图形的可见范围;如果有宽度或高度的值
//--- 分别超出源图像的宽度和高度,
//--- 则不绘制;相反,
//--- 只有对应这些值的部分才可绘制
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置将会显示在可见范围内的图像部分
//--- 默认显示图像的左上区域;该值允许
//--- 执行从该区域移动显示图像的另一个区域
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- 设置对象高亮模式启动时的边界颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置对象高亮模式启动时的边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置移动对象的定位点大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 为位图设置新图形                                                 |
//+------------------------------------------------------------------+
bool BitmapSetImage(const long   chart_ID=0,    // 图表 ID
                    const string name="Bitmap", // 位图名称
                    const string file="")       // 文件路径
  {
//--- 重置错误的值
   ResetLastError();
//--- 设置图像文件路径
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to download the image! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 在图表窗口移动位图                                               |
//+------------------------------------------------------------------+
bool BitmapMove(const long   chart_ID=0,    // 图表 ID
                const string name="Bitmap", // 位图名称
                datetime     time=0,        // 定位点时间
                double       price=0)       // 定位点价格
  {
//--- 如果没有设置点的位置，则将其移动到当前的卖价柱
   if(!time)
```

© 2000-2025, MetaQuotes Ltd.

```
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- 重置错误的值
   ResetLastError();
//--- 移动定位点
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变可见范围 (位图)大小                                          |
//+------------------------------------------------------------------+
bool BitmapChangeSize(const long   chart_ID=0,     // 图表 ID
                      const string name="Bitmap", // 位图名称
                      const int    width=0,       // 位图宽度
                      const int    height=0)      // 位图高度
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变位图大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap height! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变可见范围左上角的坐标                                         |
//+------------------------------------------------------------------+
bool BitmapMoveVisibleArea(const long   chart_ID=0,     // 图表 ID
                           const string name="Bitmap", // 位图名称
                           const int    x_offset=0,    // 可见范围 X 坐标
                           const int    y_offset=0)    // 可见范围 Y坐标<
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置错误的值
   ResetLastError();
//--- 改变位图可见范围的坐标
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Error code = ",G
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Error code = ",G
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除位图                                                         |
//+------------------------------------------------------------------+
bool BitmapDelete(const long   chart_ID=0,    // 图表 ID
                  const string name="Bitmap") // 位图名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a bitmap! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查定位点的值和为空点设置                                       |
//| 默认的值                                                         |
//+------------------------------------------------------------------+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
  {
//--- 如果点的时间没有设置，它将位于当前柱
   if(!time)
      time=TimeCurrent();
//--- 如果点的价格没有设置，则它将用卖价值
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[];  // 存储可见柱日期的数组
   double   close[]; // 存储收盘价的数组
//--- 位图文件名称
   string   file="\\Images\\dollar.bmp";
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 内存分配
   ArrayResize(date,bars);
   ArrayResize(close,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 填写收盘价的数组
   if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
     {
      Print("Failed to copy the values of Close prices! Error code = ",GetLastError())
      return;
     }
//--- 定义图像显示的频率
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- 定义步骤
   int step=1;
   switch(scale)
     {
      case 0:
         step=27;
         break;
      case 1:
         step=14;
         break;
      case 2:
         step=7;
         break;
      case 3:
         step=4;
         break;
      case 4:
         step=2;
         break;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建最高和最低柱的值的位图 间隔)。
   for(int i=0;i<bars;i+=step)
     {
      //--- 创建位图
      if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,InpWidth,InpHe
         InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
        {
         return;
        }
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 半秒延迟
   Sleep(500);
//--- 删除卖出符号
   for(int i=0;i<bars;i+=step)
     {
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_BITMAP_LABEL

位图对象标签。

注意

相对于标签的定位点位置可以从 ENUM_ANCHOR_POINT 枚举选择。定位点坐标用像素设置。

您也可以从 ENUM_BASE_CORNER 枚举选择位图定位角。

针对位图标签，您可以选择图像的 可视范围 。

示例

下面的脚本创建图表上的多个位图。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Bitmap Label\" object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string             InpName="BmpLabel";                // 标签名称
input string             InpFileOn="\\Images\\dollar.bmp"; // On 模式的文件名称
input string             InpFileOff="\\Images\\euro.bmp";  // Off 模式的文件名称
input bool               InpState=false;                    // 标签出版/发布
input ENUM_BASE_CORNER   InpCorner=CORNER_LEFT_UPPER;       // 用于定位的图表角
input ENUM_ANCHOR_POINT  InpAnchor=ANCHOR_CENTER;           // 定位类型
input color              InpColor=clrRed;                   // 高亮显示时的边界颜色
input ENUM_LINE_STYLE    InpStyle=STYLE_SOLID;              // 高亮显示时的线条风格
input int                InpPointWidth=1;                   // 移动点大小
```

© 2000-2025, MetaQuotes Ltd.

```
input bool               InpBack=false;                    // 背景对象
input bool               InpSelection=false;               // 突出移动
input bool               InpHidden=true;                   // 隐藏在对象列表
input long               InpZOrder=0;                      // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建位图标签对象                                                 |
//+------------------------------------------------------------------+
bool BitmapLabelCreate(const long              chart_ID=0,               // 图表 ID
                       const string            name="BmpLabel",          // 标签名称
                       const int               sub_window=0,             // 子窗口指数
                       const int               x=0,                      // X 坐标
                       const int               y=0,                      // Y 坐标
                       const string            file_on="",               // On 模式的图
                       const string            file_off="",              // Off 模式的
                       const int               width=0,                  // 可见范围 X
                       const int               height=0,                 // 可见范围Y坐
                       const int               x_offset=10,              // 可见范围沿着
                       const int               y_offset=10,              // 可见范围沿着
                       const bool              state=false,              // 出版/发布
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // 图表定位角
                       const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // 定位类型
                       const color             clr=clrRed,               // 高亮显示时的
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // 高亮显示时的
                       const int               point_width=1,            // 移动点大小
                       const bool              back=false,               // 在背景中
                       const bool              selection=false,          // 突出移动
                       const bool              hidden=true,              // 隐藏在对象列
                       const long              z_order=0)                // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建位图标签
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Bitmap Label\" object! Error code = ",GetLastError()
      return(false);
     }
//--- 设置On 和 Off 模式的图形
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,0,file_on))
     {
      Print(__FUNCTION__,
            ": failed to download the image for On mode! Error code = ",GetLastError()
      return(false);
     }
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,1,file_off))
     {
      Print(__FUNCTION__,
            ": failed to download the image for Off mode! Error code = ",GetLastError
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 设置标签坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置图形的可见范围;如果有宽度或高度的值
//--- 分别超出源图像的宽度和高度,
//--- 则不绘制;相反,
//--- 只有对应这些值的部分才可绘制
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置将会显示在可见范围内的图像部分
//--- 默认显示图像的左上区域;该值允许
//--- 执行从该区域移动显示图像的另一个区域
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- 定义标签状态（出版或发布）
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- 设置相对于定义点坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置定位类型
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- 设置对象高亮模式启动时的边界颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置对象高亮模式启动时的边界线条风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置移动对象的定位点大小
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 为位图标签对象设置新图形                                         |
//+------------------------------------------------------------------+
bool BitmapLabelSetImage(const long   chart_ID=0,       // 图表 ID
                         const string name="BmpLabel", // 标签名称
                         const int    on_off=0,        // 修饰符（On 或 Off）
                         const string file="")         // 文件路径
  {
//--- 重置错误的值
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//--- 设置图像文件路径
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
     {
      Print(__FUNCTION__,
            ": failed to download the image! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动位图标签对象                                                 |
//+------------------------------------------------------------------+
bool BitmapLabelMove(const long   chart_ID=0,       // 图表 ID
                     const string name="BmpLabel", // 标签名称
                     const int    x=0,             // X 坐标
                     const int    y=0)             // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动对象
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",GetLastError
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变可见范围 对象)大小                                           |
//+------------------------------------------------------------------+
bool BitmapLabelChangeSize(const long   chart_ID=0,       // 图表 ID
                           const string name="BmpLabel", // 标签名称
                           const int    width=0,         // 标签宽度
                           const int    height=0)        // 标签高度
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print(__FUNCTION__,
             ": failed to change the object width! Error code = ",GetLastError());
       return(false);
      }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
      {
       Print(__FUNCTION__,
             ": failed to change the object height! Error code = ",GetLastError());
       return(false);
      }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变可见范围左上角的坐标                                         |
//+------------------------------------------------------------------+
bool BitmapLabelMoveVisibleArea(const long   chart_ID=0,       // 图表 ID
                                const string name="BmpLabel", // 标签名称
                                const int    x_offset=0,      // 可见范围 X 坐标
                                const int    y_offset=0)      // 可见范围Y坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象可见范围的坐标
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
      {
       Print(__FUNCTION__,
             ": failed to change X coordinate of the visibility scope! Error code = ",G
       return(false);
      }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
      {
       Print(__FUNCTION__,
             ": failed to change Y coordinate of the visibility scope! Error code = ",G
       return(false);
      }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除“位图标签”对象                                               |
//+------------------------------------------------------------------+
bool BitmapLabelDelete(const long   chart_ID=0,       // 图表 ID
                       const string name="BmpLabel") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Bitmap label\" object! Error code = ",GetLastError()
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 定义位图标签坐标
   int x=(int)x_distance/2;
   int y=(int)y_distance/2;
//--- 设置标签大小和可见范围坐标
   int width=32;
   int height=32;
   int x_offset=0;
   int y_offset=0;
//--- 将位图标签置于窗口中心
   if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height,x_offset,y_
      InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidd
     {
      return;
     }
//--- 重绘图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 改变循环中标签可见范围的大小
   for(int i=0;i<6;i++)
     {
      //--- 改变可见范围大小
```

© 2000-2025, MetaQuotes Ltd.

```
      width--;
      height--;
      if(!BitmapLabelChangeSize(0,InpName,width,height))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.3 秒延迟
      Sleep(300);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 改变循环中标签可见范围的坐标
   for(int i=0;i<2;i++)
     {
      //--- 改变可见范围的坐标
      x_offset++;
      y_offset++;
      if(!BitmapLabelMoveVisibleArea(0,InpName,x_offset,y_offset))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.3 秒延迟
      Sleep(300);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除标签
   BitmapLabelDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_EDIT

编辑对象。

注意

定位点坐标用像素设置。您可以从ENUM_BASE_CORNER枚举选择编辑定位角。

您也可以从 ENUM_ALIGN_MODE 枚举在编辑中选择一种文本对齐类型。

示例

下面的脚本创建和移动图表上的编辑对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Edit\" object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string            InpName="Edit";              // 对象名称
input string            InpText="Text";              // 对象文本
input string            InpFont="Arial";             // 字体
input int               InpFontSize=14;              // 字体大小
input ENUM_ALIGN_MODE   InpAlign=ALIGN_CENTER;       // 文本对齐类型
input bool              InpReadOnly=false;           // 编辑能力
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER; // 图表定位角
input color             InpColor=clrBlack;           // 文本颜色
input color             InpBackColor=clrWhite;       // 背景色
input color             InpBorderColor=clrBlack;     // 边框颜色
input bool              InpBack=false;               // 背景对象
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpSelection=false;          // 突出移动
input bool              InpHidden=true;              // 隐藏在对象列表
input long              InpZOrder=0;                 // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建编辑对象                                                     |
//+------------------------------------------------------------------+
bool EditCreate(const long              chart_ID=0,                // 图表 ID
                const string            name="Edit",               // 对象名称
                const int               sub_window=0,              // 子窗口指数
                const int               x=0,                       // X 坐标
                const int               y=0,                       // Y 坐标
                const int               width=50,                  // 宽度
                const int               height=18,                 // 高度
                const string            text="Text",               // 文本
                const string            font="Arial",              // 字体
                const int               font_size=10,              // 字体大小
                const ENUM_ALIGN_MODE   align=ALIGN_CENTER,        // 对齐类型
                const bool              read_only=false,           // 编辑能力
                const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER,  // 图表定位角
                const color             clr=clrBlack,              // 文本颜色
                const color             back_clr=clrWhite,         // 背景色
                const color             border_clr=clrNONE,        // 边框颜色
                const bool              back=false,                // 在背景中
                const bool              selection=false,           // 突出移动
                const bool              hidden=true,               // 隐藏在对象列表
                const long              z_order=0)                 // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建编辑字段
   if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Edit\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 设置对象坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置对象大小
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置文本
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- 设置文本字体
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- 设置字体大小
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- 设置对象文本算法的类型
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- 启用 (true) 或禁用 (false) 只读模式
   ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,read_only);
//--- 设置相对于定义对象坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置文本颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置背景颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- 设置边界颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动编辑对象                                                     |
//+------------------------------------------------------------------+
bool EditMove(const long   chart_ID=0,  // 图表ID
              const string name="Edit", // 对象名称
              const int    x=0,         // X 坐标
              const int    y=0)         // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动对象
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",GetLastError
      return(false);
     }
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 调整编辑对象                                                     |
//+------------------------------------------------------------------+
bool EditChangeSize(const long   chart_ID=0,  // 图表 ID
                    const string name="Edit", // 对象名称
                    const int    width=0,     // 宽度
                    const int    height=0)    // 高度
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the object width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the object height! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变编辑对象文本                                                 |
//+------------------------------------------------------------------+
bool EditTextChange(const long   chart_ID=0,  // 图表 ID
                    const string name="Edit", // 对象名称
                    const string text="Text") // 文本
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象文本
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 返回编辑对象文本                                                 |
//+------------------------------------------------------------------+
bool EditTextGet(string      &text,        // 文本
```

© 2000-2025, MetaQuotes Ltd.

```
                  const long    chart_ID=0,  // 图表's ID
                  const string name="Edit") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 获得对象文本
   if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
     {
      Print(__FUNCTION__,
            ": failed to get the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除编辑对象                                                     |
//+------------------------------------------------------------------+
bool EditDelete(const long   chart_ID=0,  // 图表 ID
                const string name="Edit") // 对象名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Edit\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 定义改变编辑字段的步骤
   int x_step=(int)x_distance/64;
//--- 设置编辑字段坐标和其大小
   int x=(int)x_distance/8;
   int y=(int)y_distance/2;
   int x_size=(int)x_distance/8;
   int y_size=InpFontSize*2;
//--- 在本地变量存储文本
   string text=InpText;
//--- 创建编辑字段
   if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSize,InpAlign,
      InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,InpHidden,In
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 延伸编辑字段
   while(x_size-x<x_distance*5/8)
     {
      //--- increase edit field's width
      x_size+=x_step;
      if(!EditChangeSize(0,InpName,x_size,y_size))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重绘图表并等待0.05 秒
      ChartRedraw();
      Sleep(50);
     }
//--- 半秒延迟
   Sleep(500);
//--- 改变文本
   for(int i=0;i<20;i++)
     {
      //--- 在开始和结束添加 "+"
      text="+"+text+"+";
      if(!EditTextChange(0,InpName,text))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重绘图表并等待 0.1 秒
      ChartRedraw();
```

© 2000-2025, MetaQuotes Ltd.

```
      Sleep(100);
     }
//--- 半秒延迟
   Sleep(500);
//--- 删除编辑字段
   EditDelete(0,InpName);
   ChartRedraw();
//--- 等待1秒
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_EVENT

事件对象。

注意

当鼠标悬挂在事件上时，会显示其文本。

示例

下面的脚本创建和移动图表上的事件对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script draws \"Event\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="Event";     // 事件名称
input int              InpDate=25;          // 事件日期，%
input string           InpText="Text";      // 事件文本
input color            InpColor=clrRed;     // 事件颜色
input int              InpWidth=1;          // 高亮显示时的点大小
input bool             InpBack=false;       // 背景事件
input bool             InpSelection=false; // 突出移动
input bool             InpHidden=true;      // 隐藏在对象列表
input long             InpZOrder=0;         // 鼠标单击优先
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 在图表上创建事件对象                                             |
//+------------------------------------------------------------------+
bool EventCreate(const long            chart_ID=0,       // 图表 ID
                 const string          name="Event",     // 事件名称
                 const int             sub_window=0,     // 子窗口指数
                 const string          text="Text",      // 事件文本
                 datetime              time=0,           // 时间
                 const color           clr=clrRed,       // 颜色
                 const int             width=1,          // 高亮显示时的点宽度
                 const bool            back=false,       // 在背景中
                 const bool            selection=false,  // 突出移动
                 const bool            hidden=true,      // 隐藏在对象列表
                 const long            z_order=0)        // 鼠标单击优先
  {
//--- 如果没有设置时间，请在最近的柱创建对象。
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 创建事件对象
   if(!ObjectCreate(chart_ID,name,OBJ_EVENT,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Event\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 设置事件文本
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- 设置颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 如果对象突出显示则设置定位点宽度
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- 显示前景（false）或背景（true）
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用（true）或禁用（false）通过鼠标移动事件的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示（false）图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变事件对象文本                                                 |
//+------------------------------------------------------------------+
bool EventTextChange(const long   chart_ID=0,   // 图表 ID
```

© 2000-2025, MetaQuotes Ltd.

```
                     const string name="Event", // 事件名称
                     const string text="Text")  // 文本
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变对象文本
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动事件对象                                                     |
//+------------------------------------------------------------------+
bool EventMove(const long   chart_ID=0,   // 图表 ID
               const string name="Event", // 事件名称
               datetime     time=0)       // 时间
  {
//--- 如果没有设置线的时间，将事件移动到收盘柱
   if(!time)
      time=TimeCurrent();
//--- 重置错误的值
   ResetLastError();
//--- 移动对象
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move \"Event\" object! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除事件对象                                                     |
//+------------------------------------------------------------------+
bool EventDelete(const long   chart_ID=0,   // 图表 ID
                 const string name="Event") // 事件名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除对象
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
```

© 2000-2025, MetaQuotes Ltd.

```
            ": failed to delete \"Event\" object! Error code = ",GetLastError());
       return(false);
      }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查输入参数的正确性
   if(InpDate<0 || InpDate>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- 图表窗口的可见柱的数量
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- 存储要使用的日期值的数组
//--- 设置和改变事件定位点的坐标
   datetime date[];
//--- 内存分配
   ArrayResize(date,bars);
//--- 填写日期数组
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- 定义创建对象的点
   int d=InpDate*(bars-1)/100;
//--- 创建事件对象
   if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 现在，移动对象
//--- 循环计数器
   int h_steps=bars/2;
//--- 移动对象
   for(int i=0;i<h_steps;i++)
     {
      //--- 使用下面的值
```

© 2000-2025, MetaQuotes Ltd.

```
      if(d<bars-1)
         d+=1;
      //--- 移动点
      if(!EventMove(0,InpName,date[d]))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
         return;
      //--- 重画图表
      ChartRedraw();
      // 0.05 秒延迟
      Sleep(50);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 删除图表通道
   EventDelete(0,InpName);
   ChartRedraw();
//--- 1 秒延迟
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# OBJ_RECTANGLE_LABEL

矩形标签对象。

注意

定位点坐标用像素设置。您可以选择矩形标签定位角从 ENUM_BASE_CORNER 枚举。矩形标签边框类型可以从 ENUM_BORDER_TYPE 枚举来选择。

该对象被用于创建和设计自定义的图形界面。

示例

下面的脚本创建和移动图表上的矩形标签对象。已开发的特别函数用于创建和改变图形对象的属性。您可以在您的个人应用中使用这些函数"as is"。

```
//--- 描述
#property description "Script creates \"Rectangle Label\" graphical object."
//--- 启动脚本期间显示输入参数的窗口
#property script_show_inputs
//--- 脚本的输入参数
input string           InpName="RectLabel";          // 标签名称
input color            InpBackColor=clrSkyBlue;      // 背景色
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;        // 边框类型
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // 图表定位角
input color            InpColor=clrDarkBlue;         // 平面边框颜色 (Flat)
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;         // 平面边框风格 (Flat)
input int              InpLineWidth=3;               // 平面边框宽度(Flat)
input bool             InpBack=false;                // 背景对象
input bool             InpSelection=true;            // 突出移动
```

© 2000-2025, MetaQuotes Ltd.

```
input bool              InpHidden=true;              // 隐藏在对象列表
input long              InpZOrder=0;                 // 鼠标单击优先
//+------------------------------------------------------------------+
//| 创建矩形标签                                                     |
//+------------------------------------------------------------------+
bool RectLabelCreate(const long             chart_ID=0,               // 图表 ID
                     const string           name="RectLabel",         // 标签名称
                     const int              sub_window=0,             // 子窗口指数
                     const int              x=0,                      // X 坐标
                     const int              y=0,                      // Y 坐标
                     const int              width=50,                 // 宽度
                     const int              height=18,                // 高度
                     const color            back_clr=C'236,233,216',  // 背景色
                     const ENUM_BORDER_TYPE border=BORDER_SUNKEN,     // 边框类型
                     const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // 图表定位角
                     const color            clr=clrRed,               // 平面边框颜色 (
                     const ENUM_LINE_STYLE  style=STYLE_SOLID,        // 平面边框风格
                     const int              line_width=1,             // 平面边框宽度
                     const bool             back=false,               // 在背景中
                     const bool             selection=false,          // 突出移动
                     const bool             hidden=true,              // 隐藏在对象列表
                     const long             z_order=0)                // 鼠标单击优先
  {
//--- 重置错误的值
   ResetLastError();
//--- 创建矩形标签
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle label! Error code = ",GetLastError());
      return(false);
     }
//--- 设置标签坐标
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- 设置标签大小
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- 设置背景颜色
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- 设置边框类型
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- 设置相对于定义点坐标的图表的角
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- 设置平面边框颜色（在平面模式下）
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- 设置平面边框线型风格
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- 设置平面边框宽度
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- 显示前景 (false) 或背景 (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- 启用 (true) 或禁用 (false) 通过鼠标移动标签的模式
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- 在对象列表隐藏(true) 或显示 (false) 图形对象名称
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- 设置在图表中优先接收鼠标点击事件
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 移动矩形标签                                                     |
//+------------------------------------------------------------------+
bool RectLabelMove(const long   chart_ID=0,        // 图表 ID
                   const string name="RectLabel", // 标签名称
                   const int    x=0,              // X 坐标
                   const int    y=0)              // Y 坐标
  {
//--- 重置错误的值
   ResetLastError();
//--- 移动矩形标签
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",GetLastError())
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",GetLastError())
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变矩形标签的大小                                               |
//+------------------------------------------------------------------+
bool RectLabelChangeSize(const long   chart_ID=0,        // 图表 ID
                         const string name="RectLabel", // 标签名称
                         const int    width=50,         // 标签宽度
                         const int    height=18)        // 标签高度
  {
//--- 重置错误的值
   ResetLastError();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 改变标签大小
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the label's width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the label's height! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 改变矩形标签边框类型                                             |
//+------------------------------------------------------------------+
bool RectLabelChangeBorderType(const long             chart_ID=0,           // 图表 ID
                               const string           name="RectLabel",     // 标签名称
                               const ENUM_BORDER_TYPE border=BORDER_SUNKEN) // 边框类型
  {
//--- 重置错误的值
   ResetLastError();
//--- 改变边框类型
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
     {
      Print(__FUNCTION__,
            ": failed to change the border type! Error code = ",GetLastError());
      return(false);
     }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 删除矩形标签                                                     |
//+------------------------------------------------------------------+
bool RectLabelDelete(const long   chart_ID=0,       // 图表 ID
                     const string name="RectLabel") // 标签名称
  {
//--- 重置错误的值
   ResetLastError();
//--- 删除标签
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a rectangle label! Error code = ",GetLastError());
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 图表窗口大小
   long x_distance;
   long y_distance;
//--- 设置窗口大小
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- 定义矩形标签坐标
   int x=(int)x_distance/4;
   int y=(int)y_distance/4;
//--- 设置标签大小
   int width=(int)x_distance/4;
   int height=(int)y_distance/4;
//--- 创建矩形标签
   if(!RectLabelCreate(0,InpName,0,x,y,width,height,InpBackColor,InpBorder,InpCorner,
      InpColor,InpStyle,InpLineWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- 重绘图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 改变矩形标签的大小
   int steps=(int)MathMin(x_distance/4,y_distance/4);
   for(int i=0;i<steps;i++)
     {
      //--- resize
      width+=1;
      height+=1;
      if(!RectLabelChangeSize(0,InpName,width,height))
         return;
      //--- 检查脚本操作是否已经强制禁用
      if(IsStopped())
```

© 2000-2025, MetaQuotes Ltd.

```
         return;
      //--- 重绘图表并等待0.01秒
      ChartRedraw();
      Sleep(10);
     }
//--- 1 秒延迟
   Sleep(1000);
//--- 改变边框类型
   if(!RectLabelChangeBorderType(0,InpName,BORDER_RAISED))
      return;
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 改变边框类型
   if(!RectLabelChangeBorderType(0,InpName,BORDER_SUNKEN))
      return;
//--- 重画图表并等待1秒
   ChartRedraw();
   Sleep(1000);
//--- 删除标签
   RectLabelDelete(0,InpName);
   ChartRedraw();
//--- 等待1秒
   Sleep(1000);
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# 对象属性

图形对象根据对象类型可以具备多种属性。对象属性的值通过使用图形对象的相应 函数来设置和接收。

技术分析中使用的全部对象绑定时间和价格坐标：趋势线，通道，斐波纳契工具等等。但是有许多辅助对象，意在改善绑定图表始终可见部分的用户界面（主图表窗口或指标子窗口）：

| 对象 | ID | X/Y | 宽度/高度 | 日期/价格 | OBJPROP_CORNER | OBJPROP_ANCHOR | OBJPROP_ANGLE |
|---|---|---|---|---|---|---|---|
| 文本 | OBJ_TEXT | – | – | 是的 | – | 是的 | 是的 |
| 标签 | OBJ_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | 是的 |
| 按键 | OBJ_BUTTON | 是的 | 是的 | – | 是的 | – | – |
| 位图 | OBJ_BITMAP | – | 是的（只读） | 是的 | – | 是的 | – |
| 位图标签 | OBJ_BITMAP_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | – |
| 编辑 | OBJ_EDIT | 是的 | 是的 | – | 是的 | – | – |
| 矩形标签 | OBJ_RECTANGLE_LABEL | 是的 | 是的 | – | 是的 | – | – |

以下命名使用在表格中：

- **X/Y** -相对于图表角落，以像素指定的定位点坐标；
- **宽度/高度** - 对象有宽度和高度。对于“只读”，只有对象呈现在图表上时才会计算宽度和高度值；
- **日期/价格** -使用日期和价格值指定定位点坐标；
- **OBJPROP_CORNER** - 定义了指定定位点坐标相对的图表角落。可以是4个ENUM_BASE_CORNER枚举值中的一个；
- **OBJPROP_ANCHOR** -定义了对象本身的定位点，可以是9个 ENUM_ANCHOR_POINT 枚举值中的一个。从这个点到选定的图表角指定像素坐标；
- **OBJPROP_ANGLE** -定义逆时针对象旋转角度。

定义图形对象属性的函数，以及用于沿着图表创建和移动对象的ObjectCreate() 和 ObjectMove() 操作符实际上是用来向图表发送命令。如果这些函数成功执行，常见的图表事件队列就会包含该命令。当处理图表事件队列时，才会实施图形对象属性的视觉变化。

因此，不要期待调用这些函数后图形对象的即时视觉更新。通常，图表上的图形对象会根据事件变化由程序端自动更新 - 进入新报价，调整图表窗口等。使用ChartRedraw() 函数，强制更新图形对象。

© 2000-2025, MetaQuotes Ltd.

ObjectSetInteger() 和 ObjectGetInteger()函数

ENUM_OBJECT_PROPERTY_INTEGER

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| OBJPROP_COLOR | 颜色 | color |
| OBJPROP_STYLE | 类型 | ENUM_LINE_STYLE |
| OBJPROP_WIDTH | 路线厚度 | int |
| OBJPROP_BACK | 显示背景上的对象 | bool |
| OBJPROP_ZORDER | 图形对象优先接收点击图表事件(CHARTEVENT_CLICK)。创建对象时默 | long |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 认设置零值；若需要可以提高优先级。当一个接一个地应用对象时，只有级别最高的那个可以收到CHARTEVENT_CLICK事件。 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| OBJPROP_FILL | 颜色填充对象（为OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION) | bool |
| OBJPROP_HIDDEN | 禁止在程序端菜单“图表”-“对 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 象”-“对象列表”的对象列表中显示图形对象的名称。真实值允许隐藏列表的对象。默认下，true的设置用于显示日程表事 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 件的对象，交易历史和MQL5程序创建的对象。若要看到这种图形对象和访问他们的属性，请点击“对象列表”窗口中的“全部” | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 按钮。 | |
| OBJPROP_SELECTED | 所选对象 | bool |
| OBJPROP_READONLY | 在编辑对象上编辑文本 | bool |
| OBJPROP_TYPE | 对象类型 | ENUM_OBJECT r/o |
| OBJPROP_TIME | 时间坐标 | 日期时间修饰符=定位点数字 |
| OBJPROP_SELECTABLE | 对象可用性 | bool |
| OBJPROP_CREATETIME | 创建对象时间 | datetime r/o |
| OBJPROP_LEVELS | 水平数量 | int |
| OBJPROP_LEVELCOLOR | 线路水平 | 颜色修饰符=水平数字 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 颜色 | |
| OBJPROP_LEVELSTYLE | 线路水平风格 | ENUM_LINE_STYLE 修饰符=水平数字 |
| OBJPROP_LEVELWIDTH | 线路水平厚度 | int 修饰符=水平数字 |
| OBJPROP_ALIGN | “编辑”对象中的水平文本对齐(OBJ_EDIT) | ENUM_ALIGN_MODE |
| OBJPROP_FONTSIZE | 字体大小 | int |
| OBJPROP_RAY_LEFT | 向左发出射线 | bool |
| OBJPROP_RAY_RIGHT | 向右发出射线 | bool |
| OBJPROP_RAY | 垂直 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 线通过图表的全部窗口 | |
| OBJPROP_ELLIPSE | 显示裴波那契弧形对象的全部椭圆形 (OBJ_FIBOARC) | bool |
| OBJPROP_ARROWCODE | 箭头对象代码 | uchar |
| OBJPROP_TIMEFRAMES | 时间表上的对象可见性 | 设置标识 flags |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| OBJPROP_ANCHOR | 图解对象的定位点 | ENUM_ARROW_ANCHOR (用于 OBJ_ARROW), ENUM_ANCHOR_POINT (用于 OBJ_LABEL, OBJ_BITMAP_LABEL, OBJ_TEXT) |
| OBJPROP_XDISTANCE | X轴定位点像素距离 (see note) | int |
| OBJPROP_YDISTANCE | Y轴定位点像素距离 (see note) | int |
| OBJPROP_DIRECTION | 江恩物件趋势 | ENUM_GANN_DIRECTION |
| OBJPROP_DEGREE | 埃利奥特波动标记水平 | ENUM_ELLIOT_WAVE_DEGREE |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| OBJPROP_DRAWLINES | 埃利奥特波动标记排成一行 | bool |
| OBJPROP_STATE | 按钮状态（按压/未按压） | bool |
| OBJPROP_CHART_ID | “图表”对象的ID(OBJ_CHART)。它允许使用该对象的属性，好像使用图表 | long r/o |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 操作中描述的函数的普通图表，但是有一些例外。 | |
| OBJPROP_XSIZE | 对象宽度沿着X轴，以像素表示。指定用于OBJ_LABEL(只读)，OBJ_BUTTON，OBJ_CHART | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | ，OBJ_BITMAP，OBJ_BITMAP_LABEL，OBJ_EDIT，OBJ_RECTANGLE_LABEL 对象 | |
| OBJPROP_YSIZE | 对象高度沿着Y轴，以像素表示。指定用于OBJ_LABEL（只读），OBJ_BUTTON，OBJ_CH | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | ART，OBJ_BITMAP，OBJ_BITMAP_LABEL，OBJ_EDIT，OBJ_RECTANGLE_LABEL 对象 | |
| OBJPROP_XOFFSET | 图形对象“位图标签”和“位图”(OBJ_BITMAP_LABEL 和 OBJ_BITMAP)中的矩形可见 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 区域左上角的X坐标。该值由原始图片左上角对应的像素设置。 | |
| OBJPROP_YOFFSET | 图形对象“位图标签”和“位图”(OBJ_BITMAP_LABEL和OBJ_BITMAP)中的矩 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 形可见区域左上角的Y坐标。该值由原始图片左上角对应的像素设置。 | |
| OBJPROP_PERIOD | 图解对象时间表 | ENUM_TIMEFRAMES |
| OBJPROP_DATE_SCALE | 显示图解对象时间比例 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| OBJPROP_PRICE_SCALE | 显示图解对象价格比例 | bool |
| OBJPROP_CHART_SCALE | 图解对象测量 | 0-5的整数值 |
| OBJPROP_BGCOLOR | 编辑对象背景颜色OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL | color |
| OBJPROP_CORNER | 图表连接到图解 | ENUM_BASE_CORNER |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 性质类型 |
| --- | --- | --- |
| | 对象 | |
| OBJPROP_BORDER_TYPE | “矩形标签”对象的边界类型 | ENUM_BORDER_TYPE |
| OBJPROP_BORDER_COLOR | OBJ_EDIT 和 OBJ_BUTTON 对象的边界色 | color |

当使用“图表”对象(OBJ_CHART)的 图表操作 时，会有以下限制：

- 不能使用 ChartClose()关闭；
- 不能使用 ChartSetSymbolPeriod() 函数改变交易品种/周期；
- 以下属性无效 CHART_SCALE, CHART_BRING_TO_TOP, CHART_SHOW_DATE_SCALE 和 CHART_SHOW_PRICE_SCALE (ENUM_CHART_PROPERTY_INTEGER)。

对于OBJ_BITMAP_LABEL 和 OBJ_BITMAP对象，可以通过编程的方式设置特殊的图片展示模式。在该模式下，只有部分原始图片（应用矩形可见区域的图片）可以展示，而其他图片都不会显示。该区域的大小应该使用OBJPROP_XSIZE 和 OBJPROP_YSIZE属性来设置。可见区域只能使用OBJPROP_XOFFSET 和 OBJPROP_YOFFSET属性在原始图片范围内“移动”。

对于固定大小的对象： OBJ_BUTTON, OBJ_RECTANGLE_LABEL, OBJ_EDIT 和 OBJ_CHART，属性OBJPROP_XDISTANCE 和 OBJPROP_YDISTANCE 设置对象的左上角的位置到相对的图表的四角(OBJPROP_CORNER)，这里X和Y坐标将通过像素计算。

© 2000-2025, MetaQuotes Ltd.

ObjectSetDouble() 和 ObjectGetDouble() 函数

ENUM_OBJECT_PROPERTY_DOUBLE

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| OBJPROP_PRICE | 价格坐标 | double 修饰符=定位点号 |
| OBJPROP_LEVELVALUE | 水平值 | double 修饰符=水平号 |
| OBJPROP_SCALE | 比例（江恩对象和斐波纳契弧线） | double |
| OBJPROP_ANGLE | 角度。对于程序创建的，无指定角度的对象来说，它的 | double |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| | 值等同于EMPTY_VALUE | |
| OBJPROP_DEVIATION | 标准偏差通道误差 | double |

ObjectSetString() 和 ObjectGetString() 函数

ENUM_OBJECT_PROPERTY_STRING

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| OBJPROP_NAME | 物件名称 | string |
| OBJPROP_TEXT | 物件描述（物件中包含的文本） | string |
| OBJPROP_TOOLTIP | 提示信息文本。如果没有设置属性，那么将显示程序端自动生成的提示信息。提示信息可以通过设置"\n"(line feed)值来禁用 | string |
| OBJPROP_LEVELTEXT | 水平描述 | string 修饰符=水平号 |
| OBJPROP_FONT | 字体 | string |
| OBJPROP_BMPFILE | 文件的位图标签名称 另见 Resources | string 修饰符：0-state ON, 1-state OFF |
| OBJPROP_SYMBOL | 图表对象的交易品种 | string |

对于 OBJ_RECTANGLE_LABEL 对象（"矩形标签"）可以设置三个设计模式中的一种，与ENUM_BORDER_TYPE 以下值相对应。

© 2000-2025, MetaQuotes Ltd.

ENUM_BORDER_TYPE

| 标识符 | 描述 |
| --- | --- |
| BORDER_FLAT | 平面造型 |
| BORDER_RAISED | 突出造型 |
| BORDER_SUNKEN | 凹陷造型 |

对于OBJ_EDIT对象（"编辑"）和对于 ChartScreenShot() 函数，您可以使用ENUM_ALIGN_MODE枚举值来指定水平对齐类型。

ENUM_ALIGN_MODE

| 标识符 | 描述 |
| --- | --- |
| ALIGN_LEFT | 左对齐 |
| ALIGN_CENTER | 集中(仅用于编辑对象) |
| ALIGN_RIGHT | 右对齐 |

示例：

```
#define  UP          "\x0431"
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="my_OBJ_LABEL_object";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object ",label_name," not found. Error code = ",GetLastError());
      //--- 创建标签物件
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- 设置X坐标
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- 设置Y坐标
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      //--- 定义文本颜色
      ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
      //--- 定义标签物件文本
      ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
      //--- 定义字体
      ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
      //--- 定义字体大小
      ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
      //--- 顺时针45度
      ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 禁止鼠标选择
      ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);
      //--- 图表上绘画
      ChartRedraw(0);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# 对象定位方法

图形对象文本，标签，位图和位图标签（OBJ_TEXT，OBJ_LABEL，OBJ_BITMAP 和 OBJ_BITMAP_LABEL）可以有9种不同的OBJPROP_ANCHOR属性定义的坐标绑定方法中的一种。

| 对象 | ID | X/Y | 宽度/高度 | 日期/价格 | OBJPROP_CORNER | OBJPROP_ANCHOR | OBJPROP_ANGLE |
|---|---|---|---|---|---|---|---|
| 文本 | OBJ_TEXT | – | – | 是的 | – | 是的 | 是的 |
| 标签 | OBJ_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | 是的 |
| 按键 | OBJ_BUTTON | 是的 | 是的 | – | 是的 | – | – |
| 位图 | OBJ_BITMAP | – | 是的（只读） | 是的 | – | 是的 | – |
| 位图标签 | OBJ_BITMAP_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | – |
| 编辑 | OBJ_EDIT | 是的 | 是的 | – | 是的 | – | – |
| 矩形标签 | OBJ_RECTANGLE_LABEL | 是的 | 是的 | – | 是的 | – | – |

以下命名使用在表格中：

- **X/Y** - 相对于图表角落，以像素指定的定位点坐标；
- **宽度/高度** - 对象有宽度和高度。对于“只读”，只有对象呈现在图表上时才会计算宽度和高度值；
- **日期/价格** - 使用日期和价格值指定定位点坐标；
- **OBJPROP_CORNER** - 定义了指定定位点坐标相对的图表角落。可以是4个ENUM_BASE_CORNER枚举值中的一个；
- **OBJPROP_ANCHOR** - 定义了对象本身的定位点，可以是9个 ENUM_ANCHOR_POINT 枚举值中的一个。从这个点到选定的图表角指定像素坐标；
- **OBJPROP_ANGLE** - 定义逆时针对象旋转角度。

该必要的变量可以使用 ObjectSetInteger函数（chart_handle，object_name， OBJPROP_ANCHOR，anchor_point_mode）来指定，在这里anchor_point_mode是ENUM_ANCHOR_POINT值中的一个。

**ENUM_ANCHOR_POINT**

| ID | 描述 |
|---|---|
| ANCHOR_LEFT_UPPER | 左上角定位点 |
| ANCHOR_LEFT | 左侧定位点 |
| ANCHOR_LEFT_LOWER | 左下角定位点 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| ANCHOR_LOWER | 下方定位点 |
| ANCHOR_RIGHT_LOWER | 右下方定位点 |
| ANCHOR_RIGHT | 右侧定位点 |
| ANCHOR_RIGHT_UPPER | 右上角定位点 |
| ANCHOR_UPPER | 上方定位点 |
| ANCHOR_CENTER | 对象中间严格定位点 |

OBJ_BUTTON, OBJ_RECTANGLE_LABEL, OBJ_EDIT 和 OBJ_CHART 对象在左上角(ANCHOR_LEFT_UPPER)有一个修正的定位点。

示例：

```
string text_name="my_OBJ_TEXT_object";
if(ObjectFind(0,text_name)<0)
  {
   Print("Object ",text_name," not found. Error code = ",GetLastError());
   //--- 获得图表最大价格
   double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
   //--- 创建物件标签
   ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
   //--- 设置文本颜色
   ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
   //--- 设置背景色
   ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
   //--- 设置标签物件文本
   ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent()));
   //--- 设置文本字体
   ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
   //--- 设置字体大小
   ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
   //--- 定位右上角
   ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
   //--- 顺时针方向旋转90度
   ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
   //--- 禁止鼠标选择物件
   ObjectSetInteger(0,text_name,OBJPROP_SELECTABLE,false);
   //--- 重画物件
   ChartRedraw(0);
  }
```

图解对象箭头(OBJ_ARROW)连接到坐标有两种方式，标识符连接到ENUM_ARROW_ANCHOR。

**ENUM_ARROW_ANCHOR**

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| ANCHOR_TOP | 上部定位点 |
| ANCHOR_BOTTOM | 下部定位点 |

示例：

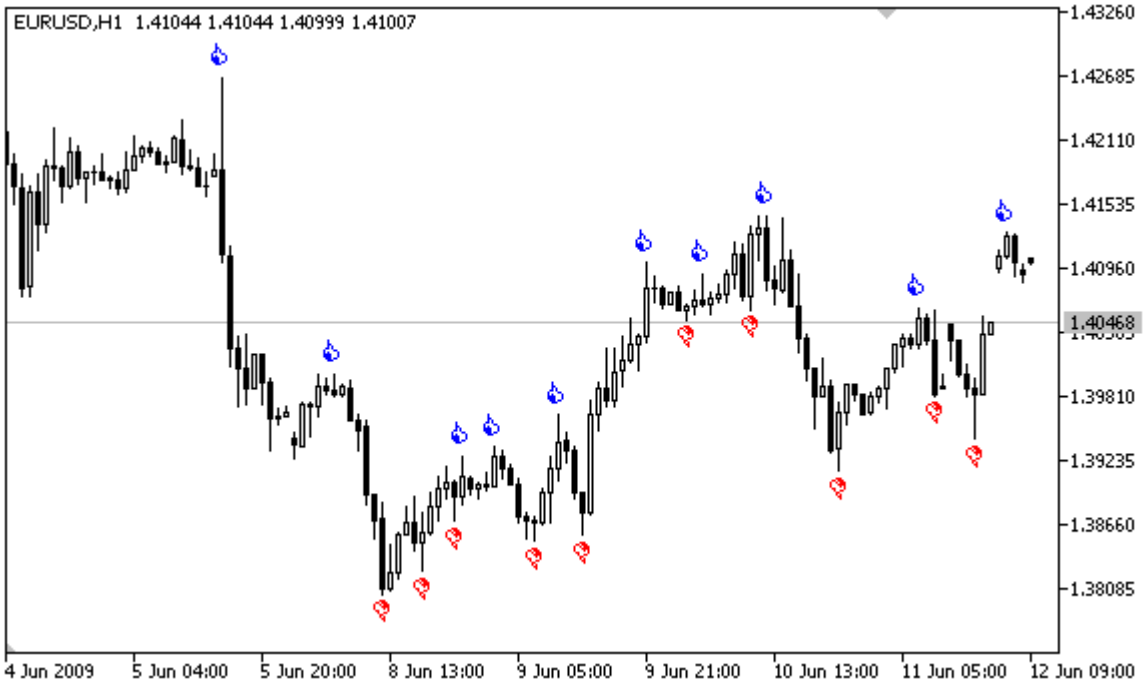
```
void OnStart()
  {
//--- 辅助数组
   double Ups[],Downs[];
   datetime Time[];
//--- 设置数组为时间序列
   ArraySetAsSeries(Ups,true);
   ArraySetAsSeries(Downs,true);
   ArraySetAsSeries(Time,true);
//--- 创建指标iFractals句柄
   int FractalsHandle=iFractals(NULL,0);
   Print("FractalsHandle = ",FractalsHandle);
//--- 设置最近的错误值到零
   ResetLastError();
//--- 复制指标值
   int copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Print("Unable to copy the upper fractals. Error = ",GetLastError());
      return;
     }

   ResetLastError();
//--- 复制指标值
   copied=CopyBuffer(FractalsHandle,1,0,1000,Downs);
   if(copied<=0)
     {
      Print("Unable to copy the bottom fractals. Error = ",GetLastError());
      return;
     }

   ResetLastError();
//--- 复制包括最近1000开盘柱的时间序列
   copied=CopyTime(NULL,0,0,1000,Time);
   if(copied<=0)
     {
      Print("Unable to copy the Opening Time of the last 1000 bars");
      return;
     }

   int upcounter=0,downcounter=0; // 数箭头数
   bool created;// 接收尝试结果去创建物件
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=2;i<copied;i++)// 运行通过指标iFractals值
     {
      if(Ups[i]!=EMPTY_VALUE)// 找到上面的分数据
        {
         if(upcounter<10)// 创建少于10个 "向上"箭头
           {
            //--- 创建 "向上" 物件
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_UP,0,Time[i],Ups[i]
            if(created)// 若设置 - 调节它
              {
               //--- 定位点在下面为了不覆盖柱
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_BOTTOM);
               //--- 最后接触-绘画
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrBlue);
               upcounter++;
              }
           }
        }
      if(Downs[i]!=EMPTY_VALUE)// 发现较低的分数据
        {
         if(downcounter<10)// 创建少于10个 "向下"箭头
           {
            //--- 创建 "向下" 物件
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_DOWN,0,Time[i],Down
            if(created)// 若设置 - 调节它
              {
               //--- 定位点在上面为了不覆盖柱
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_TOP);
               //--- 最后接触-绘画
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrRed);
               downcounter++;
              }
           }
        }
     }
  }
```

在脚本执行完毕后，图表看起来像这样。

© 2000-2025, MetaQuotes Ltd.




© 2000-2025, MetaQuotes Ltd.

# 附加物件的图表角落

这里有大量的图形对象，为此您可以设置一个图表角落，相对于它指定像素坐标。以下是对象类型（括号中是指定的对象类型标识符）：

- 标签（OBJ_LABEL）；
- 按键（OBJ_BUTTON）；
- 位图标签（OBJ_BITMAP_LABEL）；
- 编辑（OBJ_EDIT）。
- 矩形标签（OBJ_RECTANGLE_LABEL）；

| 对象 | ID | X/Y | 宽度/高度 | 日期/价格 | OBJPROP_CORNER | OBJPROP_ANCHOR | OBJPROP_ANGLE |
|---|---|---|---|---|---|---|---|
| 文本 | OBJ_TEXT | – | – | 是的 | – | 是的 | 是的 |
| 标签 | OBJ_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | 是的 |
| 按键 | OBJ_BUTTON | 是的 | 是的 | – | 是的 | – | – |
| 位图 | OBJ_BITMAP | – | 是的（只读） | 是的 | – | 是的 | – |
| 位图标签 | OBJ_BITMAP_LABEL | 是的 | 是的（只读） | – | 是的 | 是的 | – |
| 编辑 | OBJ_EDIT | 是的 | 是的 | – | 是的 | – | – |
| 矩形标签 | OBJ_RECTANGLE_LABEL | 是的 | 是的 | – | 是的 | – | – |

以下命名使用在表格中：

- **X/Y** -相对于图表角落，以像素指定的定位点坐标；
- **宽度/高度** - 对象有宽度和高度。对于“只读”，只有对象呈现在图表上时才会计算宽度和高度值；
- **日期/价格** -使用日期和价格值指定定位点坐标；
- **OBJPROP_CORNER** - 定义了指定定位点坐标相对的图表角落。可以是4个ENUM_BASE_CORNER枚举值中的一个；
- **OBJPROP_ANCHOR** -定义了对象本身的定位点，可以是9个 ENUM_ANCHOR_POINT 枚举值中的一个。从这个点到选定的图表角指定像素坐标；
- **OBJPROP_ANGLE** -定义逆时针对象旋转角度。

为了指定图表内角，X和Y坐标都要测量像素，使用 ObjectSetInteger(chartID,name,OBJPROP_CORNER,chart_corner)：

- 图表ID-图表标识符；

© 2000-2025, MetaQuotes Ltd.

- 名字 -图解对象名称；
- OBJPROP_CORNER -为捆绑指定一角的性质ID；
- **chart_corner** -目标图表，可以是 ENUM_BASE_CORNER 计算式中的一个。

**ENUM_BASE_CORNER**

| ID | 描述 |
|---|---|
| CORNER_LEFT_UPPER | 中央坐标在图表左上角 |
| CORNER_LEFT_LOWER | 中央左边在图表左下角 |
| CORNER_RIGHT_LOWER | 中央坐标在图表右下角 |
| CORNER_RIGHT_UPPER | 中央左边在图表右上角 |

示例：

```
void CreateLabel(long   chart_id,
                 string name,
                 int    chart_corner,
                 int    anchor_point,
                 string text_label,
                 int    x_ord,
                 int    y_ord)
  {
//---
   if(ObjectCreate(chart_id,name,OBJ_LABEL,0,0,0))
     {
      ObjectSetInteger(chart_id,name,OBJPROP_CORNER,chart_corner);
      ObjectSetInteger(chart_id,name,OBJPROP_ANCHOR,anchor_point);
      ObjectSetInteger(chart_id,name,OBJPROP_XDISTANCE,x_ord);
      ObjectSetInteger(chart_id,name,OBJPROP_YDISTANCE,y_ord);
      ObjectSetString(chart_id,name,OBJPROP_TEXT,text_label);
     }
   else
      Print("Failed to create the object OBJ_LABEL ",name,", Error code = ", GetLastEr
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int height=(int)ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=(int)ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   string arrows[4]={"LEFT_UPPER","RIGHT_UPPER","RIGHT_LOWER","LEFT_LOWER"};
   CreateLabel(0,arrows[0],CORNER_LEFT_UPPER,ANCHOR_LEFT_UPPER,arrows[0],50,50);
   CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,ANCHOR_RIGHT_UPPER,arrows[1],50,50);
   CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,ANCHOR_RIGHT_LOWER,arrows[2],50,50);
   CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,ANCHOR_LEFT_LOWER,arrows[3],50,50);
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# 对象可见性

对象组合可见标记决定图表时间表，该对象是明显的，为了建立或者得到OBJPROP_TIMEFRAMES类型值，可以使用 ObjectSetInteger()/ObjectGetInteger() 函数。

| ID | 值 | 描述 |
|---|---|---|
| OBJ_NO_PERIODS | 0 | 时间表中不引用对象 |
| OBJ_PERIOD_M1 | 0x00000001 | 1分钟图表显示 |
| OBJ_PERIOD_M2 | 0x00000002 | 2分钟图表显示 |
| OBJ_PERIOD_M3 | 0x00000004 | 3分钟图表显示 |
| OBJ_PERIOD_M4 | 0x00000008 | 4分钟图表显示 |
| OBJ_PERIOD_M5 | 0x00000010 | 5分钟图表显示 |
| OBJ_PERIOD_M6 | 0x00000020 | 6分钟图表显示 |
| OBJ_PERIOD_M10 | 0x00000040 | 10分钟图表显示 |
| OBJ_PERIOD_M12 | 0x00000080 | 12分钟图表显示 |
| OBJ_PERIOD_M15 | 0x00000100 | 15分钟图表显示 |
| OBJ_PERIOD_M20 | 0x00000200 | 20分钟图表显示 |
| OBJ_PERIOD_M30 | 0x00000400 | 30分钟图表显示 |
| OBJ_PERIOD_H1 | 0x00000800 | 1小时图表显示 |
| OBJ_PERIOD_H2 | 0x00001000 | 2小时图表显示 |
| OBJ_PERIOD_H3 | 0x00002000 | 3小时图表显示 |
| OBJ_PERIOD_H4 | 0x00004000 | 4小时图表显示 |
| OBJ_PERIOD_H6 | 0x00008000 | 6小时图表显示 |
| OBJ_PERIOD_H8 | 0x00010000 | 8小时图表显示 |
| OBJ_PERIOD_H12 | 0x00020000 | 12小时图表显示 |
| OBJ_PERIOD_D1 | 0x00040000 | 以天为单位的图表显示 |
| OBJ_PERIOD_W1 | 0x00080000 | 以周为单位的图表显示 |
| OBJ_PERIOD_MN1 | 0x00100000 | 以月为单位的图表显示 |
| OBJ_ALL_PERIODS | 0x001fffff | 所有时间表对象显示 |

可见性标记可以通过使用符号"|"来建立，例如，标记组合OBJ_PERIOD_M10|OBJ_PERIOD_H4表示该对象在10分钟到4小时这个时间段内可见

示例：

```
void OnStart()
  {

```

© 2000-2025, MetaQuotes Ltd.

```
//---
   string highlevel="PreviousDayHigh";
   string lowlevel="PreviousDayLow";
   double prevHigh;           // 前一天最高价
   double prevLow;            // 前一天最低价
   double highs[],lows[];     // 最高和最低价数组

//--- 重置上一个错误
   ResetLastError();
//--- 获得日常时间表最近的2个最高值
   int highsgot=CopyHigh(Symbol(),PERIOD_D1,0,2,highs);
   if(highsgot>0) // 如果复制成功
     {
      Print("High prices for the last 2 days were obtained successfully");
      prevHigh=highs[0]; // 前一天最高价
      Print("prevHigh = ",prevHigh);
      if(ObjectFind(0,highlevel)<0) // 没有找到名为highlevel物件
        {
         ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // 创建水平线物件
        }
      //--- 为高位线价格平面设置值
      ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
      //--- 仅设置PERIOD_M10和PERIOD_H4的可见性
      ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else
     {
      Print("Could not get High prices over the past 2 days, Error = ",GetLastError());
     }

//--- 重置上一个错误
   ResetLastError();
//--- 获得日常时间表最近的2个最低值
   int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2,lows);
   if(lowsgot>0) // 如果复制成功
     {
      Print("Low prices for the last 2 days were obtained successfully");
      prevLow=lows[0]; // 前一天最低价
      Print("prevLow = ",prevLow);
      if(ObjectFind(0,lowlevel)<0) // 没有找到名为lowlevel物件
        {
         ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // 创建水平线物件
        }
      //--- 为低位线价格平面设置值
      ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
      //--- 仅设置PERIOD_M10和PERIOD_H4的可见性
      ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else Print("Could not get Low prices for the last 2 days, Error = ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartRedraw(0); // 强制重画图表
  }
```

另见

秒周期，周期，图表时间表， 日期和时间

© 2000-2025, MetaQuotes Ltd.

# 埃利奥特波动水平

埃利奥特波动是由两个图解对象类型OBJ_ELLIOTWAVE5和OBJ_ELLIOTWAVE3表示，为创建波动大小（波动标签方法），使用OBJPROP_DEGREE 属性，计算式ENUM_ELLIOT_WAVE_DEGREE中的值可以表示出来。

**ENUM_ELLIOT_WAVE_DEGREE**

| ID | 描述 |
|---|---|
| ELLIOTT_GRAND_SUPERCYCLE | 庞大的超周期 |
| ELLIOTT_SUPERCYCLE | 超周期 |
| ELLIOTT_CYCLE | 循环 |
| ELLIOTT_PRIMARY | 原色 |
| ELLIOTT_INTERMEDIATE | 媒介 |
| ELLIOTT_MINOR | 次要的 |
| ELLIOTT_MINUTE | 分钟 |
| ELLIOTT_MINUETTE | 波浪 |
| ELLIOTT_SUBMINUETTE | 潜波浪 |

示例：

```
   for(int i=0;i<ObjectsTotal(0);i++)
     {
      string currobj=ObjectName(0,i);
      if((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE3) ||
          ((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE5)))
        {
         //--- 在INTERMEDIATE设置标记水平
         ObjectSetInteger(0,currobj,OBJPROP_DEGREE,ELLIOTT_INTERMEDIATE);
         //--- 显示波峰间的线
         ObjectSetInteger(0,currobj,OBJPROP_DRAWLINES,true);
         //--- 设置线颜色
         ObjectSetInteger(0,currobj,OBJPROP_COLOR,clrBlue);
         //--- 设置线宽
         ObjectSetInteger(0,currobj,OBJPROP_WIDTH,5);
         //--- 设置描述
         ObjectSetString(0,currobj,OBJPROP_TEXT,"test script");
        }
     }
```

© 2000-2025, MetaQuotes Ltd.

# 江恩物件

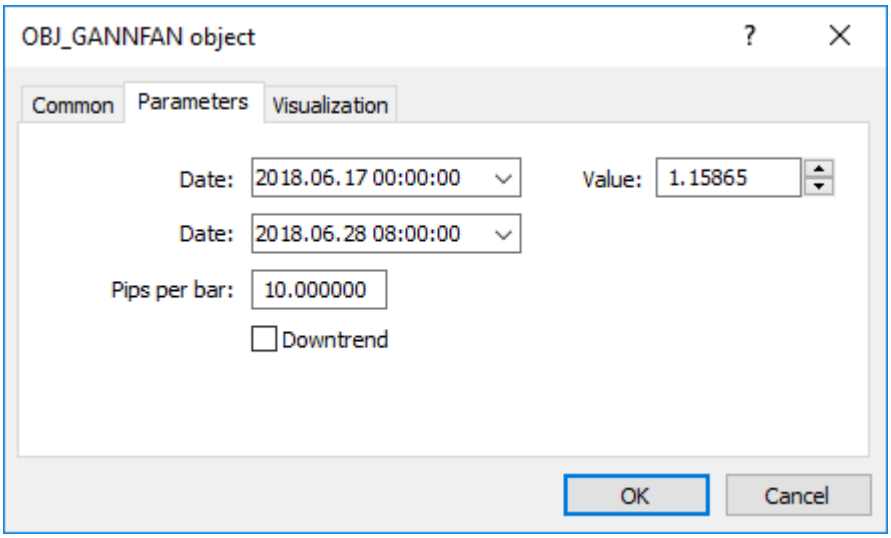
对于江恩角度线（OBJ_GANNFAN）和江恩网格（OBJ_GANNGRID）来说，ENUM_GANN_DIRECTION 中的两种值可以指定出来，会在开头建立。

**ENUM_GANN_DIRECTION**

| ID | 描述 |
|---|---|
| GANN_UP_TREND | 与上升趋势一致的线 |
| GANN_DOWN_TREND | 与下降趋势一致的线 |

建立主干线1x1规模，使用 ObjectSetDouble 函数(chart_handle, gann_object_name, OBJPROP_SCALE, scale):

- chart_handle -对象加载的图表窗口；
- gann_object_name -对象名称；
- OBJPROP_SCALE -"Scale"属性标识符；
- 比例 -需要以Pips/Bar为单位。




创建江恩角度线示例：

```
void OnStart()
  {
//---
   string my_gann="OBJ_GANNFAN object";
   if(ObjectFind(0,my_gann)<0)//物件没找到
     {
      //--- 通知失败
      Print("Object ",my_gann," not found. Error code = ",GetLastError());
      //--- 获得图表最大价格
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- 获得图表最小价格
      double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
      //--- 多少柱显示在图表上?
```

© 2000-2025, MetaQuotes Ltd.

```
      int bars_on_chart=ChartGetInteger(0,CHART_VISIBLE_BARS);
      //--- 创建一个数组，写下每个柱的开盘时间
      datetime Time[];
      //--- 像时间序列一样访问数组
      ArraySetAsSeries(Time,true);
      //--- 现在复制图表中可见的柱数据到数组中
      int times=CopyTime(NULL,0,0,bars_on_chart,Time);
      if(times<=0)
        {
         Print("Could not copy the array with the open time!");
         return;
        }
      //--- 完成预备

      //--- 图表中心柱指数
      int center_bar=bars_on_chart/2;
      //--- 图表赤道 - 最大和最小值之间
      double mean=(chart_max_price+chart_min_price)/2.0;
      //--- 设置第一定位点坐标到中心
      ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Time[center_bar],mean,
                   //--- 第二定位点到右边
                   Time[center_bar/2],(mean+chart_min_price)/2.0);
      Print("Time[center_bar] = "+(string)Time[center_bar]+"  Time[center_bar/2] = "+
      //Print("Time[center_bar]/="+Time[center_bar]+"  Time[center_bar/2]="+Time[cente
      //--- 设置 Pips / Bar单元比例
      ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
      //--- 设置趋向线
      ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
      //--- 设置线宽
      ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
      //--- 定义线型
      ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
      //--- 设置线的颜色
      ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
      //--- 允许用户选择物件
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
      //--- 自动选择
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
      //--- 图表上绘画
      ChartRedraw(0);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# 网页颜色

以下颜色参量是由颜色类型定义的：

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| clrBlack | clrDarkGreen | clrDarkSlateGray | clrOlive | clrGreen | clrTeal | clrNavy | clrPurple |
| clrMaroon | clrIndigo | clrMidnightBlue | clrDarkBlue | clrDarkOliveGreen | clrSaddleBrown | clrForestGreen | clrOliveDrab |
| clrSeaGreen | clrDarkGoldenrod | clrDarkSlateBlue | clrSienna | clrMediumBlue | clrBrown | clrDarkTurquoise | clrDimGray |
| clrLightSeaGreen | clrDarkViolet | clrFireBrick | clrMediumVioletRed | clrMediumSeaGreen | clrChocolate | clrCrimson | clrSteelBlue |
| clrGoldenrod | clrMediumSpringGreen | clrLawnGreen | clrCadetBlue | clrDarkOrchid | clrYellowGreen | clrLimeGreen | clrOrangeRed |
| clrDarkOrange | clrOrange | clrGold | clrYellow | clrChartreuse | clrLime | clrSpringGreen | clrAqua |
| clrDeepSkyBlue | clrBlue | clrMagenta | clrRed | clrGray | clrSlateGray | clrPeru | clrBlueViolet |
| clrLightSlateGray | clrDeepPink | clrMediumTurquoise | clrDodgerBlue | clrTurquoise | clrRoyalBlue | clrSlateBlue | clrDarkKhaki |
| clrIndianRed | clrMediumOrchid | clrGreenYellow | clrMediumAquamarine | clrDarkSeaGreen | clrTomato | clrRosyBrown | clrOrchid |
| clrMediumPurple | clrPaleVioletRed | clrCoral | clrCornflowerBlue | clrDarkGray | clrSandyBrown | clrMediumSlateBlue | clrTan |
| clrDarkSalmon | clrBurlyWood | clrHotPink | clrSalmon | clrViolet | clrLightCoral | clrSkyBlue | clrLightSalmon |
| clrPlum | clrKhaki | clrLightGreen | clrAquamarine | clrSilver | clrLightSkyBlue | clrLightSteelBlue | clrLightBlue |
| clrPaleGreen | clrThistle | clrPowderBlue | clrPaleGoldenrod | clrPaleTurquoise | clrLightGray | clrWheat | clrNavajoWhite |
| clrMoccasin | clrLightPink | clrGainsboro | clrPeachPuff | clrPink | clrBisque | clrLightGoldenrod | clrBlanchedAlmond |
| clrLemonChiffon | clrBeige | clrAntiqueWhite | clrPapayaWhip | clrCornsilk | clrLightYellow | clrLightCyan | clrLinen |
| clrLavender | clrMistyRose | clrOldLace | clrWhiteSmoke | clrSeashell | clrIvory | clrHoneydew | clrAliceBlue |
| clrLavenderBlush | clrMintCream | clrSnow | clrWhite | | | | |

通过使用 ObjectSetInteger() 函数，颜色可以建立一个对象，使用 PlotIndexSetInteger() 函数，建立颜色到常规指标中。为了得到颜色值，还可以使用简单函数 ObjectGetInteger() 和 PlotIndexGetInteger().

© 2000-2025, MetaQuotes Ltd.

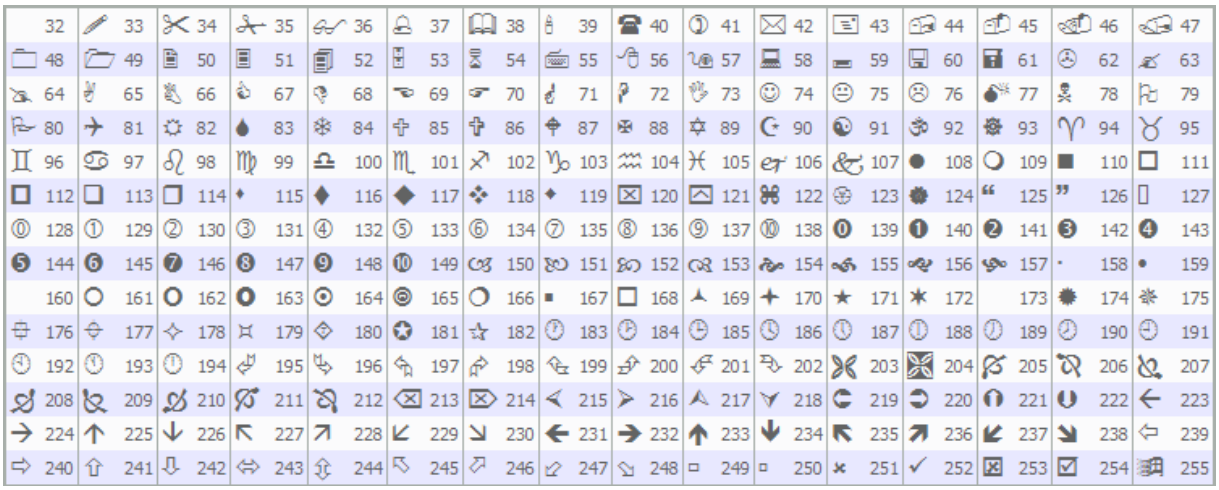
示例：

```
//---- 指标设置
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_type3   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_color2  clrRed
#property indicator_color3  clrLime
```

© 2000-2025, MetaQuotes Ltd.

## Wingdings

该字符使用 OBJ_ARROW 对象：




通过使用 ObjectSetInteger() 函数，可以建立必要字符。

示例：

```
void OnStart()
  {
//---
   string up_arrow="up_arrow";
   datetime time=TimeCurrent();
   double lastClose[1];
   int close=CopyClose(Symbol(),Period(),0,1,lastClose);      // 获得收盘价
//--- 如果获得价格
   if(close>0)
     {
      ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0,0);           // 创建一个箭头
      ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);     // 设置箭头代码
      ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);         // 设置时间
      ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);// 预定价格
      ChartRedraw(0);                                         // 现在绘制箭头
     }
   else
      Print("Unable to get the latest Close price!");
  }
```

© 2000-2025, MetaQuotes Ltd.

# 指标常量

预定义37个 技术指标 ，可以使用在MQL5语言程序中，此外，它为使用 iCustom() 函数创建自定义指标提供了机会，所有常量都被分成5组：

- 价格常量 -选择价格和成交量类型，指标是可以评估的；
- 平滑方式 -指标中使用的内置滤波方式；
- 指标线 -当使用 CopyBuffer() 访问指标值时的指标缓冲区标识符；
- 绘画样式 -建立并书写18字节的线条设计；
- 自定义指标属性 与 自定义 指标工作时使用在函数中；
- 指标类型 用来当使用IndicatorCreate()建立处理器时指明技术指标；
- 数据类型标识符 用来当把MqlParam数组传到IndicatorCreate()函数中时指明数据类型。

© 2000-2025, MetaQuotes Ltd.

# 价格常数

技术指标的计算要求价格价值和成交量价值，该计算将执行，有7个预先的标识符在ENUM_APPLIED_PRICE项目中，以指定计算的期望价值基础。

ENUM_APPLIED_PRICE

| ID | 描述 |
|---|---|
| PRICE_CLOSE | 收盘价格 |
| PRICE_OPEN | 开盘价格 |
| PRICE_HIGH | 一个时期的最高价格 |
| PRICE_LOW | 一个时期的最低价格 |
| PRICE_MEDIAN | 中间值 (高+低)/2 |
| PRICE_TYPICAL | 典型价格 (高+低+收盘价)/3 |
| PRICE_WEIGHTED | 平均价格 (高+低+收盘价格+开盘价格)/4 |

如果成交量用于计算，必要在ENUM_APPLIED_VOLUME中列举出一两个值

ENUM_APPLIED_VOLUME

| ID | 描述 |
|---|---|
| VOLUME_TICK | 赊欠成交量 |
| VOLUME_REAL | 交易成交量 |

iStochastic()技术指标以两种方式计算使用：

- 或者只以收盘价；
- 或者以最高或最低价；

为计算选择必须变量，指定ENUM_STO_PRICE计算中的一种值

ENUM_STO_PRICE

| ID | 描述 |
|---|---|
| STO_LOWHIGH | 基于最低价/最高价的计算 |
| STO_CLOSECLOSE | 基于开盘价/收盘价的计算 |

如果技术指标使用价格数据的计算，建立ENUM_APPLIED_PRICE类型，然后任何指标处理 (内置在终端里或者使用者输入)都能用来输入价格序列。在此情况下，指标中零缓冲器的值会用来计算，这样就很容易使用另一指标建造该指标的值，自定义指标的处理需要调用 iCustom() 函数处理。

示例：

```
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 输入函数
input int      RSIperiod=14;          // 计算RSI的周期
input int      Smooth=8;              // 平滑RSI周期
input ENUM_MA_METHOD meth=MODE_SMMA; // 修匀法
//---- 图 RSI
#property indicator_label1  "RSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//---- 图 RSI_Smoothed
#property indicator_label2  "RSI_Smoothed"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrNavy
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 指标缓冲区
double         RSIBuffer[];           // 这里存储 RSI值
double         RSI_SmoothedBuffer[]; //  RSI平滑值
int            RSIhandle;             // 处理 RSI 指标
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图indicator buffers mapping
   SetIndexBuffer(0,RSIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,RSI_SmoothedBuffer,INDICATOR_DATA);
   IndicatorSetString(INDICATOR_SHORTNAME,"iRSI");
   IndicatorSetInteger(INDICATOR_DIGITS,2);
//---
   RSIhandle=iRSI(NULL,0,RSIperiod,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[]
                )

  {
//---  重置上一个错误值
   ResetLastError();
//--- 数组RSIBuffer []中获得 RSI 指标数据
   int copied=CopyBuffer(RSIhandle,0,0,rates_total,RSIBuffer);
   if(copied<=0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Unable to copy the values of the indicator RSI. Error = ",
             GetLastError(),",  copied =",copied);
       return(0);
      }
//--- 创建使用RSI值的平均值的指标
   int RSI_MA_handle=iMA(NULL,0,Smooth,0,meth,RSIhandle);
   copied=CopyBuffer(RSI_MA_handle,0,0,rates_total,RSI_SmoothedBuffer);
   if(copied<=0)
     {
      Print("Unable to copy the smoothed indicator of RSI. Error = ",
            GetLastError(),",  copied =",copied);
      return(0);
     }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平滑方式

许多技术指标都是以各种价位序列平滑方式为基础的，许多标准技术指标要求输入平滑型的规格作为输入参数，对于平滑型描述类型的说明来说，可使用ENUM_MA_METHOD列表中的标识符。

ENUM_MA_METHOD

| ID | 描述 |
| --- | --- |
| MODE_SMA | 简单平均数 |
| MODE_EMA | 指数平均数 |
| MODE_SMMA | 平滑平均数 |
| MODE_LWMA | 线性平均数 |

示例：

```
double ExtJaws[];
double ExtTeeth[];
double ExtLips[];
//---- 移动平均数句柄
int    ExtJawsHandle;
int    ExtTeethHandle;
int    ExtLipsHandle;
//--- 获得 MA's 句柄
ExtJawsHandle=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtTeethHandle=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtLipsHandle=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
```

© 2000-2025, MetaQuotes Ltd.

# 指标线

在图表中许多技术指标会显示几个缓冲区，指标缓冲区起始数字是0，当使用CopyBuffer()函数复制指标值到双精度类型数组时，对于一些指标来说，它可以代替数字表示复制标识符。

当复制 iMACD(), iRVI() 和 iStochastic() 值时可用的指标线标识符

| Constant | 值 | 描述 |
|---|---|---|
| MAIN_LINE | 0 | 干线 |
| SIGNAL_LINE | 1 | 信号线 |

当复制 ADX() 和 ADXW() 值时可用的指标线标识符。

| 常量 | 值 | 描述 |
|---|---|---|
| MAIN_LINE | 0 | 干线 |
| PLUSDI_LINE | 1 | 线 +DI |
| MINUSDI_LINE | 2 | 线 -DI |

当复制iBands()值时可用的指标线标识符

| 常量 | 值 | 描述 |
|---|---|---|
| BASE_LINE | 0 | 干线 |
| UPPER_BAND | 1 | 最高限制 |
| LOWER_BAND | 2 | 最低限制 |

当复制 iEnvelopes() 和 iFractals()值时可用的指标线标识符

| 常量 | 值 | 描述 |
|---|---|---|
| UPPER_LINE | 0 | 上边线 |
| LOWER_LINE | 1 | 下边线 |

当复制iGator()值时可用的指标线标识符。

| 常量 | 值 | 描述 |
|---|---|---|
| UPPER_HISTOGRAM | 0 | 上直方图 |
| LOWER_HISTOGRAM | 2 | 底部直方图 |

当复制 iAlligator()值时可用的指标线标识符。

| 常量 | 值 | 描述 |
|---|---|---|
| GATORJAW_LINE | 0 | 爪线 |
| GATORTEETH_LINE | 1 | 牙线 |
| GATORLIPS_LINE | 2 | 唇线 |

© 2000-2025, MetaQuotes Ltd.

当复制iIchimoku()值时可用的指标线标识符。

| 常量 | 值 | 描述 |
| --- | --- | --- |
| TENKANSEN_LINE | 0 | 转换线 |
| KIJUNSEN_LINE | 1 | 基准线 |
| SENKOUSPANA_LINE | 2 | 闪光跨度A线 |
| SENKOUSPANB_LINE | 3 | 闪光跨度B线 |
| CHIKOUSPAN_LINE | 4 | 延迟线 |

© 2000-2025, MetaQuotes Ltd.

# 绘画风格

当创建一个自定义指标，可以指定18字节的图解测绘中任意一种（显示在基本图表窗口或者图表子窗口），其价格指定在ENUM_DRAW_TYPE计算式中。

在一个自定义指标中，可以使用任意指标新建/绘画风格，为存储必要数据，每个建筑风格要求5种全局数组数组中的一个。通过使用SetIndexBuffer()函数可使这些数据数组与指标缓冲区绑定在一起，ENUM_INDEXBUFFER_TYPE数据类型是每个缓冲区指定的。

依据绘图风格，需要1-4个缓冲区（显示为INDICATOR_DATA）。如果该风格承认兼容的动态交替颜色（所有包含在颜色中的名称），就需要更多颜色的缓冲区（指示类型INDICATOR_COLOR_INDEX）。在价值缓冲区与风格相一致时，颜色缓冲区会受到限制。

ENUM_DRAW_TYPE

| ID | 描述 | 数据缓冲 | 颜色缓冲 |
|---|---|---|---|
| DRAW_NONE | 没有画线 | 1 | 0 |
| DRAW_LINE | 画线 | 1 | 0 |
| DRAW_SECTION | 线条 | 1 | 0 |
| DRAW_HISTOGRAM | 从0位线画柱状图 | 1 | 0 |
| DRAW_HISTOGRAM2 | 两个指标命令的柱状图 | 2 | 0 |
| DRAW_ARROW | 画箭头 | 1 | 0 |
| DRAW_ZIGZAG | 之字设计允许垂直界面 | 2 | 0 |
| DRAW_FILLING | 两层间的颜色 | 2 | 0 |
| DRAW_BARS | 以字符序列展示 | 4 | 0 |
| DRAW_CANDLES | 以蜡烛台序列展示 | 4 | 0 |
| DRAW_COLOR_LINE | 多色线 | 1 | 1 |
| DRAW_COLOR_SECTION | 多色分段 | 1 | 1 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 数据缓冲 | 颜色缓冲 |
|---|---|---|---|
| DRAW_COLOR_HISTOGRAM | 从0线开始的多色直方图 | 1 | 1 |
| DRAW_COLOR_HISTOGRAM2 | 两个指标缓冲区的多色直方图 | 2 | 1 |
| DRAW_COLOR_ARROW | 描绘多色箭头 | 1 | 1 |
| DRAW_COLOR_ZIGZAG | 多色之字形 | 2 | 1 |
| DRAW_COLOR_BARS | 多色字节 | 4 | 1 |
| DRAW_COLOR_CANDLES | 多色蜡烛台 | 4 | 1 |

提取所选描绘类型标识符，ENUM_PLOT_PROPERTY列表列出函数 PlotIndexSetInteger() 和 PlotIndexGetInteger()

**ENUM_PLOT_PROPERTY_INTEGER**

| ID | 描述 | 属性类型 |
|---|---|---|
| PLOT_ARROW | DRAW_ARROW类型箭头代码 | uchar |
| PLOT_ARROW_SHIFT | DRAW_ARROW类型垂直箭头转换 | int |
| PLOT_DRAW_BEGIN | 在DataWindow中没有描绘和值的原始字节数量 | int |
| PLOT_DRAW_TYPE | 图解建筑类型 | ENUM_DRAW_TYPE |
| PLOT_SHOW_DATA | 在DataWindow中显示建筑值标志 | bool |
| PLOT_SHIFT | 在字节中与时间轴一起描绘的指标转换 | int |
| PLOT_LINE_STYLE | 画线类型 | ENUM_LINE_STYLE |
| PLOT_LINE_WIDTH | 画线层次 | int |
| PLOT_COLOR_INDEXES | 颜色数量 | int |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| PLOT_LINE_COLOR | 包含绘画颜色的缓冲区指标 | 颜色 修饰符 = 颜色数量指标 |

函数 PlotIndexSetDouble()

ENUM_PLOT_PROPERTY_DOUBLE

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| PLOT_EMPTY_VALUE | 测绘空值，没有绘图 | double |

函数 PlotIndexSetString()

ENUM_PLOT_PROPERTY_STRING

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| PLOT_LABEL | 显示在DataWindow中的指标图解系列的名称. 当使用复杂的需要多个指标缓冲区来显示的图形风格时，可以使用","作为分隔符来指定每个缓冲区的名称。示例代码显示在DRAW_CANDLES | string |

在自定义指标中，有5种类型可以使用绘画线条，他们只在或者厚度时是有效的

ENUM_LINE_STYLE

| ID | 描述 |
| --- | --- |
| STYLE_SOLID | 实线 |
| STYLE_DASH | 折线 |
| STYLE_DOT | 虚线 |
| STYLE_DASHDOT | 折点线 |
| STYLE_DASHDOTDOT | 双折点线 |

为了建立线条画类型和描绘类型，可以使用 PlotIndexSetInteger() 函数，使用函数 ObjectSetInteger() 可以标明斐波纳契引申厚度和描绘水平类型。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 指标缓冲区
double         MABuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 绑定数组和指数为0的指标缓冲区
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
//--- 设置画线
   PlotIndexSetInteger(0,PLOT_DRAW_TYPE,DRAW_LINE);
//--- 设置线型
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,STYLE_DOT);
//--- 设置线颜色
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrRed);
//--- 设置线粗细
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
//--- 为线设置标签
   PlotIndexSetString(0,PLOT_LABEL,"Moving Average");
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   for(int i=prev_calculated;i<rates_total;i++)
     {
      MABuffer[i]=close[i];
     }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 自定义指标属性

用在自定义指标中的指标缓冲区的数量是受限的，使用SetIndexBuffer()函数可以设计成自定义缓冲区，因为要存储，所有指定数据类型是有必要的，也可能是ENUM_INDEXBUFFER_TYPE其中一个值。

**ENUM_INDEXBUFFER_TYPE**

| ID | 描述 |
|---|---|
| INDICATOR_DATA | 描绘数据 |
| INDICATOR_COLOR_INDEX | 颜色 |
| INDICATOR_CALCULATIONS | 媒介运算的辅助缓冲区 |

自定义指标提供许多设置提供方便演绎，这些设置通过使用IndicatorSetDouble(),IndicatorSetInteger()和 IndicatorSetString() 函数来分别相应指标性质，指标属性的标识符列表在ENUM_CUSTOMIND_PROPERTY中。

**ENUM_CUSTOMIND_PROPERTY_INTEGER**

| ID | 描述 | 属性类型 |
|---|---|---|
| INDICATOR_DIGITS | 描绘指标值的精确度 | int |
| INDICATOR_HEIGHT | 修正指标窗口的高度(预处理命令#property indi | int |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| | cator_height) | |
| INDICATOR_LEVELS | 指标窗口中的水平数量 | int |
| INDICATOR_LEVELCOLOR | 水平线颜色 | color 修饰符 = 层号 |
| INDICATOR_LEVELSTYLE | 水平线类型 | ENUM_LINE_STYLE 修饰符 = 层号 |
| INDICATOR_LEVELWIDTH | 水平线厚度 | int 修饰符 = 层号 |
| INDICATOR_FIXED_MINIMUM | 修正指标窗口最小值。该属性只能通过 | bool |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| | IndicatorSetInteger()函数写入 | |
| INDICATOR_FIXED_MAXIMUM | 修正指标窗口最大值。该属性只能通过IndicatorSetInteger()函数写入 | bool |

ENUM_CUSTOMIND_PROPERTY_DOUBLE

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| INDICATOR_MINIMUM | 指标窗口最小化 | double |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| INDICATOR_MAXIMUM | 指标窗口最大化 | double |
| INDICATOR_LEVELVALUE | 水平值 | double 修饰符 = 层号 |

ENUM_CUSTOMIND_PROPERTY_STRING

| ID | 描述 | 属性类型 |
| --- | --- | --- |
| INDICATOR_SHORTNAME | 短指标名称 | string |
| INDICATOR_LEVELTEXT | 水平描述 | string 修饰符=层号 |

示例：

```
//--- 指标设置
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_color2  clrRed
//--- 输入参量
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//--- 指标缓冲区
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,MainBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,HighesBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,LowesBuffer,INDICATOR_CALCULATIONS);
//--- 设置精确性
   IndicatorSetInteger(INDICATOR_DIGITS,2);
//--- 设置水平
   IndicatorSetInteger(INDICATOR_LEVELS,2);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,0,20);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,1,80);
//--- 为子窗口设置最大最小值
   IndicatorSetDouble(INDICATOR_MINIMUM,0);
   IndicatorSetDouble(INDICATOR_MAXIMUM,100);
//--- 设置第一条画指数的柱
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,KPeriod+Slowing-2);
   PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,KPeriod+Slowing+DPeriod);
//--- 为第二行设置STYLE_DOT类型
   PlotIndexSetInteger(1,PLOT_LINE_STYLE,STYLE_DOT);
//--- 为DataWindow和指标子窗口标签命名
   IndicatorSetString(INDICATOR_SHORTNAME,"Stoch("+KPeriod+","+DPeriod+","+Slowing+")"
   PlotIndexSetString(0,PLOT_LABEL,"Main");
   PlotIndexSetString(1,PLOT_LABEL,"Signal");
//--- 设置画线空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0.0);
//--- 完成初始化
  }
```

© 2000-2025, MetaQuotes Ltd.

# 技术指标类型

为使用 接入值，有两种方法建立指标处理器，第一种方法是从技术指标 列表中直接指定函数名称，第二种方法是使用IndicatorCreate()，通过分配ENUM_INDICATOR中的标识符共同增添任意指标处理。两种处理办法的创建结果是相同的，当使用MQL5编辑程序时，在特定情况下使用最简便的。

当创建指标IND_CUSTOM类型时，输入参数 MqlParam 数组中第一个元素的类型领域一定有ENUM_DATATYPE，表达式中的 TYPE_STRING 值，而第一个元素中的*string_value*一定包括自定义指标中的名称。

**ENUM_INDICATOR**

| 标识符 | 指标 |
|---|---|
| IND_AC | 加速振荡器 |
| IND_AD | 聚集/分散 |
| IND_ADX | 均定向指标 |
| IND_ADXW | 亚当理论的ADX |
| IND_ALLIGATOR | 鳄鱼指标 |
| IND_AMA | 相应的移动平均数 |
| IND_AO | 动量振荡指标-AO指标 |
| IND_ATR | 真实波动幅度均值 |
| IND_BANDS | 布林线指标 (AO指标) |
| IND_BEARS | 熊市 |
| IND_BULLS | 牛市 |
| IND_BWMFI | 市场便利指标 |
| IND_CCI | 顺势指标 |
| IND_CHAIKIN | 佳庆指标 |
| IND_CUSTOM | 自定义指标 |
| IND_DEMA | 双精度移动平均线 |
| IND_DEMARKER | DEM指标 |
| IND_ENVELOPES | 轨道线指标 |
| IND_FORCE | 强力指数 |
| IND_FRACTALS | 拼图 |
| IND_FRAMA | 自适拼图移动平均数 |
| IND_GATOR | 振荡器 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 指标 |
| --- | --- |
| IND_ICHIMOKU | 一目均衡图 |
| IND_MA | 平均移动 |
| IND_MACD | MACD |
| IND_MFI | 货币流量索引 |
| IND_MOMENTUM | 动量 |
| IND_OBV | 平衡成交量 |
| IND_OSMA | OsMA |
| IND_RSI | 相对强势索引 |
| IND_RVI | 相对活力索引 |
| IND_SAR | 抛物线SAR |
| IND_STDDEV | 标准偏差 |
| IND_STOCHASTIC | 随机振荡器 |
| IND_TEMA | 三倍指数移动平均值 |
| IND_TRIX | 三倍指数移动平均值振荡器 |
| IND_VIDYA | 索引变量平均值 |
| IND_VOLUMES | 成交量 |
| IND_WPR | 威廉姆斯百分比幅度 |

© 2000-2025, MetaQuotes Ltd.

# 数据类型标识符

当使用IndicatorCreate()函数创建自定义处理器时，MqlParam 类型数组会以最后一个参量规定，因此，MqlParam 结构，描述的指标，包括一种特殊的字段类型。该领域包括的关于数据类型 (实数, 整数 或者 字符串 类型)的信息是通过数组中的特殊元素传递的，MqlParam 结构中的值或许是ENUM_DATATYPE值中的一个。

**ENUM_DATATYPE**

| 标识符 | 数据类型 |
|---|---|
| TYPE_BOOL | bool |
| TYPE_CHAR | char |
| TYPE_UCHAR | uchar |
| TYPE_SHORT | short |
| TYPE_USHORT | ushort |
| TYPE_COLOR | color |
| TYPE_INT | int |
| TYPE_UINT | uint |
| TYPE_DATETIME | datetime |
| TYPE_LONG | long |
| TYPE_ULONG | ulong |
| TYPE_FLOAT | float |
| TYPE_DOUBLE | double |
| TYPE_STRING | string |

数组中的每一个元素都能描述创建的技术指标录入函数，因此数组中元素的类型和命令一定与描述保持严格一致。

© 2000-2025, MetaQuotes Ltd.

# 环境状态

MQL5程序中描述当前运行环境的常量分为如下组：

- 客户端属性 -客户端信息；
- 运行MQL5程序属性 -MQL5程序特征，它帮助控制执行命令；
- 交易品种属性 -有关交易品种的信息；
- 账户属性 -有关现金账户的信息；
- 测试统计 -EA交易的测试结果。

© 2000-2025, MetaQuotes Ltd.

# 客户端属性

有关客户端信息包含在两种函数中，TerminalInfoInteger()和TerminalInfoString()。对于参量来说，这些函数分别从ENUM_TERMINAL_INFO_INTEGER 和 ENUM_TERMINAL_INFO_STRING中接收值。

### ENUM_TERMINAL_INFO_INTEGER

| 标识符 | 描述 | 类型 |
|---|---|---|
| TERMINAL_BUILD | 客户端构造编号 | int |
| TERMINAL_COMMUNITY_ACCOUNT | 标识表示程序端中存在MQL5.community授权数据 | bool |
| TERMINAL_COMMUNITY_CONNECTION | 连接 MQL5.community | bool |
| TERMINAL_CONNECTED | 连接的交易服务器 | bool |
| TERMINAL_DLLS_ALLOWED | 使用DLL许可 | bool |
| TERMINAL_TRADE_ALLOWED | 允许交易 | bool |
| TERMINAL_EMAIL_ENABLED | 在制定终端允许使用SMTP-server和 login发送邮件 | bool |
| TERMINAL_FTP_ENABLED | 在制定终端允许使用FTP-server和 login发送报告 | bool |
| TERMINAL_NOTIFICATIONS_ENABLED | 允许向智能手机发送通知 | bool |
| TERMINAL_MAXBARS | 图表中的最大字节 | int |
| TERMINAL_MQID | 标识表示存在MetaQuotes ID 数据 推送通知 | bool |
| TERMINAL_CODEPAGE | 在客户端建立的 语言代码页 数字 | int |
| TERMINAL_CPU_CORES | 系统的CPU 内核数量 | int |
| TERMINAL_DISK_SPACE | 程序端 (代理)MQL5\Files 文件夹的空闲磁盘空间，MB | int |
| TERMINAL_MEMORY_PHYSICAL | 系统的物理内存，MB | int |
| TERMINAL_MEMORY_TOTAL | 程序端 (代理)进程的可用内存，MB | int |
| TERMINAL_MEMORY_AVAILABLE | 程序端 (代理)进程的空闲内存，MB | int |
| TERMINAL_MEMORY_USED | 程序端 (代理)使用的内存，MB | int |
| TERMINAL_X64 | "64-位程序端" | bool |
| TERMINAL_OPENCL_SUPPORT | OpenCL 支持的版本格式 0x00010002 = 1.2。"0" 表示OpenCL 不被支持 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| TERMINAL_SCREEN_DPI | 屏幕上显示信息的分辨率是以每英寸一行的点数计算的（DPI）。<br>知道该参数值，您可以设置图形对象的大小，以便在不同分辨率的显示器上看起来一样。 | int |
| TERMINAL_SCREEN_LEFT | 虚拟屏幕的左坐标。虚拟屏幕是覆盖所有监视器的长方形。如果系统从右至左有两个监视器，那么虚拟屏幕的左坐标可以在两个监视器的边界上。 | int |
| TERMINAL_SCREEN_TOP | 虚拟屏幕的顶部坐标 | int |
| TERMINAL_SCREEN_WIDTH | 程序端宽度 | int |
| TERMINAL_SCREEN_HEIGHT | 程序端高度 | int |
| TERMINAL_LEFT | 相对于虚拟屏幕的程序端的左坐标 | int |
| TERMINAL_TOP | 相对于虚拟屏幕的程序端的顶部坐标 | int |
| TERMINAL_RIGHT | 相对于虚拟屏幕的程序端的右坐标 | int |
| TERMINAL_BOTTOM | 相对于虚拟屏幕的程序端的底部坐标 | int |
| TERMINAL_PING_LAST | 最后知道的交易服务器的微秒ping值。一秒包含一百万微秒 | int |
| TERMINAL_VPS | 表示程序端在MetaTrader虚拟主机服务器（MetaTrader VPS）上启动 | bool |
| 按键标识符 | 描述 | |
| TERMINAL_KEYSTATE_LEFT | “左箭头”键状态 | int |
| TERMINAL_KEYSTATE_UP | “向上箭头”键状态 | int |
| TERMINAL_KEYSTATE_RIGHT | “右箭头”键状态 | int |
| TERMINAL_KEYSTATE_DOWN | “向下箭头”键状态 | int |
| TERMINAL_KEYSTATE_SHIFT | “Shift”键状态 | int |
| TERMINAL_KEYSTATE_CONTROL | “Ctrl”键状态 | int |
| TERMINAL_KEYSTATE_MENU | “Windows”键状态 | int |
| TERMINAL_KEYSTATE_CAPSLOCK | “CapsLock”键状态 | int |
| TERMINAL_KEYSTATE_NUMLOCK | “Num Lock”键状态 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| TERMINAL_KEYSTATE_SCRLOCK | “ScrollLock”键状态 | int |
| TERMINAL_KEYSTATE_ENTER | “Enter”键状态 | int |
| TERMINAL_KEYSTATE_INSERT | “Insert”键状态 | int |
| TERMINAL_KEYSTATE_DELETE | “Delete”键状态 | int |
| TERMINAL_KEYSTATE_HOME | “Home”键状态 | int |
| TERMINAL_KEYSTATE_END | “End”键状态 | int |
| TERMINAL_KEYSTATE_TAB | “Tab”键状态 | int |
| TERMINAL_KEYSTATE_PAGEUP | “PageUp”键状态 | int |
| TERMINAL_KEYSTATE_PAGEDOWN | “PageDown”键状态 | int |
| TERMINAL_KEYSTATE_ESCAPE | “Escape”键状态 | int |

调用TerminalInfoInteger(TERMINAL_KEYSTATE_XXX)返回GetKeyState()函数在MSDN相同键的状态代码。

比例系数计算示例：

```
//--- 在屏幕上创建 1.5 英寸宽的按键
int screen_dpi = TerminalInfoInteger(TERMINAL_SCREEN_DPI); // 找到用户显示器的DPI
int base_width = 144;                                      // DPI=96标准显示器的屏幕点的
int width      = (button_width * screen_dpi) / 96;         // 为用户显示器（特别DPI）计算按
...

//--- 按百分比计算比例系数
int scale_factor=(TerminalInfoInteger(TERMINAL_SCREEN_DPI) * 100) / 96;
//--- 使用比例系数
width=(base_width * scale_factor) / 100;
```

在以上示例中，图形资源在不同分辨率特征的显示器上看起来一样。控制元素的大小（按键，对话框等等）对应个性化设置。

### ENUM_TERMINAL_INFO_DOUBLE

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| TERMINAL_COMMUNITY_BALANCE | MQL5.community的结余 | 双精度 |
| TERMINAL_RETRANSMISSION | 对于所有在指定计算机上运行应用程序和服务的TCP/IP协议中的现有网络数据包的百分比。即使在最快且配置正确的网络中也可能发生数据包的丢失。在这 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 种情况下，在接收方与发送方之间没有确认数据包的交付，因此丢失的数据包被重新发送。<br><br>这并不表示特定程序端和交易服务器之间的连接质量，因为这个百分比是为整个网络活动计算，包括系统和后台活动。<br><br>TERMINAL_RETRANSMISSION值每分钟从操作系统请求一次。 程序端本身并不计算该值。 | |

只能在两种目录下执行文件操作；使用TERMINAL_DATA_PATH 和 TERMINAL_COMMONDATA_PATH 要求特征包括相似路径。

### ENUM_TERMINAL_INFO_STRING

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| TERMINAL_LANGUAGE | 程序端语言 | string |
| TERMINAL_COMPANY | 公司名称 | string |
| TERMINAL_NAME | 程序端名称 | string |
| TERMINAL_PATH | 程序端文件夹启动 | string |
| TERMINAL_DATA_PATH | 程序端数据文件夹存储 | string |
| TERMINAL_COMMONDATA_PATH | 电脑中所有程序端的普通路径 | string |
| TERMINAL_CPU_NAME | CPU名称 | string |
| TERMINAL_CPU_ARCHITECTURE | CPU架构 | string |
| TERMINAL_OS_VERSION | 用户操作系统名称 | string |
| TERMINAL_COLORTHEME_NAME | 终端配色方案；可能的值：浅色和深色。 | string |

为更好的了解路径，存储在TERMINAL_PATH，TERMINAL_DATA_PATH 和 TERMINAL_COMMONDATA_PATH参量中的属性，推荐使用脚本，它可以通过当前复制的客户端的来返回这些值。

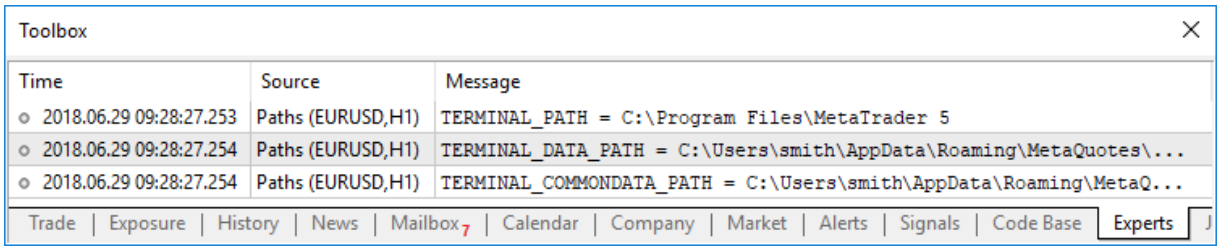
示例：脚本返回的有关客户端路径的信息

```
//+------------------------------------------------------------------+
//|                                        Check_TerminalPaths.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   Print("TERMINAL_PATH = ",TerminalInfoString(TERMINAL_PATH));
   Print("TERMINAL_DATA_PATH = ",TerminalInfoString(TERMINAL_DATA_PATH));
   Print("TERMINAL_COMMONDATA_PATH = ",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
  }
```

脚本执行的结果在Experts Journa1中可以看到相关信息，如下表示：




## 取得终端配色方案信息

终端支持两种配色方案：浅色（默认）和深色。在开发具有图形用户界面的自定义应用程序时，程序员应考虑当前的终端配色方案。应用程序中使用的视觉组件应具有动态适应性，以增强用户体验并保持视觉一致性。

为了支持这一点，本语言提供了检测终端配色方案的函数：

- ENUM_TERMINAL_INFO_STRING枚举中的 TERMINAL_COLORTHEME_NAME 值允许您通过使用 TerminalInfoString 函数来获取当前配色方案的名称。可能的值：浅色和深色。
- 使用 ENUM_TERMINAL_INFO_INTEGER枚举中的 THEME_COLOR_*值可以通过 TerminalInfoInteger 函数获取特定 UI元素的颜色。

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| THEME_COLOR_WINDOW | 窗口背景 | color |
| THEME_COLOR_WINDOWTEXT | 窗口中的文本 | color |
| THEME_COLOR_BTNTEXT | 按钮文本 | color |
| THEME_COLOR_GRAYTEXT | 非活动（被禁用）的文本 | color |
| THEME_COLOR_INFOTEXT | 提示文本 | color |
| THEME_COLOR_INFOBK | 提示背景 | color |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| THEME_COLOR_3DFACE | 3D 元素正面 | color |
| THEME_COLOR_3DLIGHT | 3D 元素的光侧 | color |
| THEME_COLOR_3DSHADOW | 3D 元素的阴影侧 | color |
| THEME_COLOR_3DDKSHADOW | 3D 元素的阴影 | color |
| THEME_COLOR_3DHILIGHT | 3D 元素的高亮显示 | color |
| THEME_COLOR_HIGHLIGHT | 选定元素的背景 | color |
| THEME_COLOR_HIGHLIGHTTEXT | 选定元素的文本 | color |
| THEME_COLOR_BTNFACE | 按钮正面 | color |
| THEME_COLOR_BTNHILIGHT | 按钮高亮显示 | color |
| THEME_COLOR_BTNSHADOW | 按钮阴影 | color |
| THEME_COLOR_MENU | 菜单背景 | color |
| THEME_COLOR_MENUBAR | 菜单栏背景 | color |
| THEME_COLOR_MENUTEXT | 菜单文本 | color |
| THEME_COLOR_MENUHILIGHT | 选中菜单项的高亮显示 | color |
| THEME_COLOR_ACTIVECAPTION | 活动窗口标题 | color |
| THEME_COLOR_INACTIVECAPTION | 非活动窗口标题 | color |
| THEME_COLOR_GRADIENTINACTIVECAPTION | 非活动窗口标题的渐变 | color |
| THEME_COLOR_CAPTIONTEXT | 窗口标题文本 | color |
| THEME_COLOR_INACTIVECAPTIONTEXT | 非活动窗口标题文本 | color |
| THEME_COLOR_HOTTEXT | 活动元素的超链接 | color |
| THEME_COLOR_NONE | 未选择颜色 | color |
| THEME_COLOR_SEPARATOR | 分隔线 | color |
| THEME_COLOR_SCROLLBACK | 滚动条 | color |
| THEME_COLOR_LINE1 | 日记中奇数行的背景颜色 | color |
| THEME_COLOR_LINE2 | 日记中偶数行的背景颜色 | color |
| THEME_COLOR_GRID | 日志中的网格颜色 | color |
| THEME_COLOR_SUMMARY | 日志中摘要行的背景颜色 | color |
| THEME_COLOR_ERROR | 错误消息文本颜色 | color |
| THEME_COLOR_INVALID | 无效值文本的颜色 | color |
| THEME_COLOR_NEGATIVE | 负值颜色 | color |
| THEME_COLOR_POSITIVE | 正值颜色 | color |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| THEME_COLOR_LINK | 链接颜色 | color |
| THEME_COLOR_LINKHOVER | 链接悬停颜色 | color |
| THEME_COLOR_LINKTESTER | 来自缓存的先前测试/优化运行的结果中的链接颜色 | color |
| THEME_COLOR_TEXTUP | “按钮释放”状态 | color |
| THEME_COLOR_TEXTDOWN | “按钮按下”状态 | color |
| THEME_COLOR_BACKUP | 报价升高时“一键交易”中“买入”和“卖出”按钮的颜色 | color |
| THEME_COLOR_BACKDOWN | 报价下降时“一键交易”中“买入”和“卖出”按钮的颜色 | color |
| THEME_COLOR_CLOSE | “关闭”操作按钮颜色 | color |
| THEME_COLOR_BUY | “买入”操作按钮颜色 | color |
| THEME_COLOR_SELL | “卖出”操作按钮颜色 | color |
| THEME_COLOR_DEPOSIT | “存款”按钮颜色 | color |
| THEME_COLOR_WITHDRAWAL | “提款”按钮颜色 | color |
| THEME_COLOR_BID | 买价线颜色 | color |
| THEME_COLOR_ASK | 卖价线颜色 | color |
| THEME_COLOR_STOPS | 止损线颜色 | color |
| THEME_COLOR_STOPS_RED | 当利润为负时止损值高亮显示（交易选项卡） | color |
| THEME_COLOR_STOPS_GREEN | 当利润为正时止损值高亮显示（交易选项卡） | color |
| THEME_COLOR_CONFIRM | 订单提交窗口中“接受”按钮颜色 | color |
| THEME_COLOR_REQUOTE | 订单提交窗口中“重新报价”按钮颜色 | color |
| THEME_COLOR_REJECT | 订单提交窗口中“拒绝”按钮颜色 | color |
| THEME_COLOR_NOTIFICATION | 订单提交窗口中来自服务器的更改通知的颜色 | color |
| THEME_COLOR_RATING | 学习系统中的评分栏颜色 | color |
| THEME_COLOR_BOOK_BUY | 市场深度中买入水平的背景颜色 | color |
| THEME_COLOR_BOOK_SELL | 市场深度中卖出水平的背景颜色 | color |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 属性类型 |
| --- | --- | --- |
| THEME_COLOR_BOOK_LAST | 市场深度中最后交易的颜色 | color |
| THEME_COLOR_BOOK_STOP | 市场深度中的止损水平颜色 | color |
| THEME_COLOR_BOOK_SPREAD | 市场深度价差内水平的背景颜色 | color |
| THEME_COLOR_TICKS_BID | 订单提交窗口中分时图上的买价线颜色 | color |
| THEME_COLOR_TICKS_ASK | 订单提交窗口中分时图上的卖价线颜色 | color |
| THEME_COLOR_TICKS_LAST | 订单提交窗口中分时图上的最新交易线颜色 | color |
| THEME_COLOR_TICKS_CROSS | 订单提交窗口中分时图上的十字线颜色 | color |
| THEME_COLOR_TICKS_SL | 订单提交窗口中分时图上的止损线颜色 | color |
| THEME_COLOR_TICKS_TP | 订单提交窗口中分时图上的止盈线颜色 | color |
| THEME_COLOR_TESTER_START | 测试/优化中的“开始”按钮颜色 | color |
| THEME_COLOR_TESTER_STOP | 测试/优化中的“停止”按钮颜色 | color |
| THEME_COLOR_TESTER_START_FRAME | 测试/优化中“开始”按钮的边框颜色 | color |
| THEME_COLOR_TESTER_STOP_FRAME | 测试/优化中“停止”按钮的边框颜色 | color |
| THEME_COLOR_TESTER_PROGRESS | 测试/优化中的进度条颜色 | color |
| THEME_COLOR_TESTER_BALANCE | 策略测试器中的余额线颜色 | color |
| THEME_COLOR_TESTER_EQUITY | 策略测试器中的净值线颜色 | color |
| THEME_COLOR_TESTER_MARGIN | 策略测试器中的存款负载图颜色 | color |
| THEME_COLOR_PROFILER_CALL | 分析期间调用的代码行的颜色 | color |
| THEME_COLOR_PROFILER_CALLSEL | 分析期间调用时选择的代码行颜色 | color |
| THEME_COLOR_PROFILER_LINE | 分析日志中的线条颜色 | color |
| THEME_COLOR_PROFILER_LINESEL | 分析日志中选定的线条颜色 | color |

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# 运行MQL程序属性

为了获取当前运行的MQL5程序的信息，可以使用ENUM_MQL_INFO_INTEGER 和 ENUM_MQL_INFO_STRING常量。

函数 MQLInfoInteger

**ENUM_MQL_INFO_INTEGER**

| 标识符 | 描述 | 类型 |
|---|---|---|
| MQL_HANDLES_USED | 当前活动对象句柄的数量。这些包括动态（通过new值创建）和非动态对象、全局/局部变量或类成员。程序使用的句柄越多，它消耗的资源就越多 | int |
| MQL_MEMORY_LIMIT | MQL5程序最大可能的动态内存数，MB计算 | int |
| MQL_MEMORY_USED | MQL5程序使用的内存大小，MB计算 | int |
| MQL_PROGRAM_TYPE | MQL5程序类型 | ENUM_PROGRAM_TYPE |
| MQL_DLLS_ALLOWED | 允许为已生效的程序交易 | bool |
| MQL_TRADE_ALLOWED | 允许为已生效的程序交易 | bool |
| MQL_SIGNALS_ALLOWED | 允许为已生效的程序更改信号 | bool |
| MQL_DEBUG | 表示程序在调试模式下运行 | bool |
| MQL_PROFILER | 表示程序在代码分析模式下运行 | bool |
| MQL_TESTER | 表示程序在测试中运行 | bool |
| MQL_FORWARD | 表示程序在前向测试过程中运行 | bool |
| MQL_OPTIMIZATION | 表示程序在优化模式下运行 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| MQL_VISUAL_MODE | 表示程序在可视测试模式下运行 | bool |
| MQL_FRAME_MODE | 表示EA交易在收集优化结果框架模式下运行 | bool |
| MQL_LICENSE_TYPE | EX5模块的许可证类型。许可证涉及EX5模块，从这里使用MQLInfoInteger(MQL_LICENSE_TYPE)发出请求。 | ENUM_LICENSE_TYPE |
| MQL_STARTED_FROM_CONFIG | 如果脚本/EA是从配置文件的启动(StartUp)部分启动，则返回true。这意味着该脚本/EA已在启动程序端的配置文件中注册。 | bool |

函数 MQLInfoString

**ENUM_MQL_INFO_STRING**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| MQL_PROGRAM_NAME | mql5已执行程序名称 | string |
| MQL_PROGRAM_PATH | 以执行系统路径 | string |

运行程序类型信息，使用ENUM_PROGRAM_TYPE值

**ENUM_PROGRAM_TYPE**

| 标识符 | 描述 |
| --- | --- |
| PROGRAM_SCRIPT | 脚本 |
| PROGRAM_EXPERT | 专家 |
| PROGRAM_INDICATOR | 指标 |
| PROGRAM_SERVICE | 服务 |

**ENUM_LICENSE_TYPE**

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| LICENSE_FREE | 免费无限使用版 |
| LICENSE_DEMO | 市场付费产品的试用版仅在策略测试中工作 |
| LICENSE_FULL | 购买的授权版允许至少5次激活。激活次数由卖家设定。卖家可以提高允许的激活次数 |
| LICENSE_TIME | 有期限限制的授权版 |

示例：

```
ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PROGRAM_TYPE);
switch(mql_program)
  {
   case PROGRAM_SCRIPT:
     {
      Print(__FILE__+" is script");
      break;
     }
   case PROGRAM_EXPERT:
     {
      Print(__FILE__+" is Expert Advisor");
      break;
     }
   case PROGRAM_INDICATOR:
     {
      Print(__FILE__+" is custom indicator");
      break;
     }
   default:Print("MQL5 type value is ",mql_program);
//---
Print("MQLInfoInteger(MQL_MEMORY_LIMIT)=", MQLInfoInteger(MQL_MEMORY_LIMIT), " MB")
Print("MQLInfoInteger(MQL_MEMORY_USED)=", MQLInfoInteger(MQL_MEMORY_USED), " MB");
Print("MQLInfoInteger(MQL_HANDLES_USED)=", MQLInfoInteger(MQL_HANDLES_USED), " hanc
```

© 2000-2025, MetaQuotes Ltd.

# 交易品种属性

可以通过 SymbolInfoInteger()，SymbolInfoDouble() 和 SymbolInfoString() 函数获得当前市场信息。第一个参量是交易品种名称，第二个函数参量的值是ENUM_SYMBOL_INFO_INTEGER, ENUM_SYMBOL_INFO_DOUBLE 和 ENUM_SYMBOL_INFO_STRING标识符中的一个。

函数 SymbolInfoInteger()

**ENUM_SYMBOL_INFO_INTEGER**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_SECTOR | 资产所属的经济部门 | ENUM_SYMBOL_SECTOR |
| SYMBOL_INDUSTRY | 交易品种所属的行业或经济分支 | ENUM_SYMBOL_INDUSTRY |
| SYMBOL_CUSTOM | 这是一个自定义交易品种该交易 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 品种是基于来自市场报价和/或外部数据源的另一个交易品种而综合创建的 | |
| SYMBOL_BACKGROUND_COLOR | 市场报价中的交易品种所使用的背 | color |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 景色 | |
| SYMBOL_CHART_MODE | 生成交易品种柱形图所使用的价格类型，例如卖价或最后价 | ENUM_SYMBOL_CHART_MODE |
| SYMBOL_EXIST | 同名交易品种已存在 | bool |
| SYMBOL_SELECT | 在市场报价中选择交易 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 品种 | |
| SYMBOL_VISIBLE | 交易品种在市场报价中可见。<br><br>一些交易品种（主要是计算入金货币预付款要求或利润所需的交叉率）被自动选 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 择，但通常在市场报价中不可见。若要显示这些交易品种，就需要选择显式显示。 | |
| SYMBOL_SESSION_DEALS | 当前时期的交易数量 | long |
| SYMBOL_SESSION_BUY_ORDERS | 当时的买入订 | long |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 单数量 | |
| SYMBOL_SESSION_SELL_ORDERS | 当时的卖出订单数量 | long |
| SYMBOL_VOLUME | 最后订单成交量 | long |
| SYMBOL_VOLUMEHIGH | 当天最大订单 | long |
| SYMBOL_VOLUMELOW | 当天最小订单 | long |
| SYMBOL_TIME | 最后报价时间 | datetime |
| SYMBOL_TIME_MSC | 自1970.01.01后的最后 | long |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 报价时间（以毫秒计算） | |
| SYMBOL_DIGITS | 小数点后数字 | int |
| SYMBOL_SPREAD_FLOAT | 浮点传播迹象 | bool |
| SYMBOL_SPREAD | 相关传播值 | int |
| SYMBOL_TICKS_BOOKDEPTH | 显示在市场深度要求中的最大数量，交易品种无 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 队列要求，值是0 | |
| SYMBOL_TRADE_CALC_MODE | 合约价格计算方式 | ENUM_SYMBOL_CALC_MODE |
| SYMBOL_TRADE_MODE | 订单执行类型 | ENUM_SYMBOL_TRADE_MODE |
| SYMBOL_START_TIME | 交易品种交易开始日期（通常用于期货） | datetime |
| SYMBOL_EXPIRATION_TIME | 交易品种交易结束日期 | datetime |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | （通常用于期货） | |
| SYMBOL_TRADE_STOPS_LEVEL | 止蚀盘当前收盘价格的最小空间 | int |
| SYMBOL_TRADE_FREEZE_LEVEL | 凝结交易操作的距离 | int |
| SYMBOL_TRADE_EXEMODE | 订单执行方式 | ENUM_SYMBOL_TRADE_EXECUTION |
| SYMBOL_SWAP_MODE | 交易计算模式 | ENUM_SYMBOL_SWAP_MODE |
| SYMBOL_SWAP_ROLLOVER3DAYS | 日翻滚 | ENUM_DAY_OF_WEEK |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 收费 | |
| SYMBOL_MARGIN_HEDGED_USE_LEG | 计算锁仓预付款只收取单边交易手数大的（买入或卖出） | bool |
| SYMBOL_EXPIRATION_MODE | 到期模式允许命令标志 | int |
| SYMBOL_FILLING_MODE | 填充模式允许命令标志 | int |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_ORDER_MODE | 允许的标识订单类型 | int |
| SYMBOL_ORDER_GTC_MODE | 止损和盈利订单到期模式，如果SYMBOL_EXPIRATION_MODE=SYMBOL_EXPIRATION_GTC（有效直至被取消） | ENUM_SYMBOL_ORDER_GTC_MODE |
| SYMBOL_OPTION_MODE | 期权类型 | ENUM_SYMBOL_OPTION_MODE |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_OPTION_RIGHT | 期权权限（看涨/看跌） | ENUM_SYMBOL_OPTION_RIGHT |

函数 SymbolInfoDouble()

**ENUM_SYMBOL_INFO_DOUBLE**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_BID | 买价—最佳卖出信息 | 双精度 |
| SYMBOL_BIDHIGH | 一天中最高买价 | 双精度 |
| SYMBOL_BIDLOW | 一天中最低买价 | 双精度 |
| SYMBOL_ASK | 卖价—最佳买入信息 | 双精度 |
| SYMBOL_ASKHIGH | 一天中最高买价 | 双精度 |
| SYMBOL_ASKLOW | 一天中最低买价 | 双精度 |
| SYMBOL_LAST | 最后订单价格 | 双精度 |
| SYMBOL_LASTHIGH | 一天中最高 | 双精度 |
| SYMBOL_LASTLOW | 一天中最低 | 双精度 |
| SYMBOL_VOLUME_REAL | 最后订单成交量 | 双精度 |
| SYMBOL_VOLUMEHIGH_REAL | 当天最大订单 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_VOLUMELOW_REAL | 当天最小订单 | 双精度 |
| SYMBOL_OPTION_STRIKE | 期权的执行价。期权买家可以购买（看涨期权）或出售（看跌期权）基础资产的价格，期权卖家有义务出售或购买基础资产的适当数量。 | 双精度 |
| SYMBOL_POINT | 交易品种点值 | 双精度 |
| SYMBOL_TRADE_TICK_VALUE | SYMBOL_TRADE_TICK_VALUE_PROFIT 的值 | 双精度 |
| SYMBOL_TRADE_TICK_VALUE_PROFIT | 为盈利持仓计算报价 | 双精度 |
| SYMBOL_TRADE_TICK_VALUE_LOSS | 为亏损持仓计算报价 | 双精度 |
| SYMBOL_TRADE_TICK_SIZE | 最小价格改变 | 双精度 |
| SYMBOL_TRADE_CONTRACT_SIZE | 交易合约大小 | 双精度 |
| SYMBOL_TRADE_ACCRUED_INTEREST | 应计利息 - 累积的票 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 面利率，例如，部分票面利率是按照付息债权发行的天数或最后一次支付票面利率的天数来计算的 | |
| SYMBOL_TRADE_FACE_VALUE | 票面价值 -发行人设置的初始债券价值 | 双精度 |
| SYMBOL_TRADE_LIQUIDITY_RATE | 流动性利率是可用于预付款的资产份额。 | 双精度 |
| SYMBOL_VOLUME_MIN | 一笔订单中的最小成交量 | 双精度 |
| SYMBOL_VOLUME_MAX | 一笔订单中的最大成交量 | 双精度 |
| SYMBOL_VOLUME_STEP | 交易执行的最小成交量更改步骤 | 双精度 |
| SYMBOL_VOLUME_LIMIT | 交易品种单向（单买或单卖）持仓和挂 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 单可允许的最大总容量。例如，5手限定范围内，您可以拥有5手买入持仓交易量和下单5手卖出限价挂单交易量。但是在这种情况下，你不能下单买入限价挂单（因为单向总交易量将会超出限定范围）或下单超过5手的卖出限价交易量。 | |
| SYMBOL_SWAP_LONG | 买入库存费值 | 双精度 |
| SYMBOL_SWAP_SHORT | 卖出库存费值 | 双精度 |
| SYMBOL_SWAP_SUNDAY | 周日到次日隔夜持仓的库存费计算比率 (SYMBOL_SWAP | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | _LONG 或 SYMBOL_SWAP_SHORT)。支持以下值：<br>• 0 - 不收取库存费<br>• 1 - 单次库存费<br>• 3 - 三倍库存费 | |
| SYMBOL_SWAP_MONDAY | 周一到周二隔夜持仓的库存费计算比率 (SYMBOL_SWAP_LONG 或 SYMBOL_SWAP_SHORT) | 双精度 |
| SYMBOL_SWAP_TUESDAY | 周二到周三隔夜持仓的库存 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 费计算比率(SYMBOL_SWAP_LONG或SYMBOL_SWAP_SHORT) | |
| SYMBOL_SWAP_WEDNESDAY | 周三到周四隔夜持仓的库存费计算比率(SYMBOL_SWAP_LONG或SYMBOL_SWAP_SHORT) | 双精度 |
| SYMBOL_SWAP_THURSDAY | 周四到周五隔夜持仓的库存费计算比率(SYMBOL_SWAP_LONG或SYMBOL_SWAP_SHORT) | 双精度 |
| SYMBOL_SWAP_FRIDAY | 周五到周六隔夜持仓的库存费计算比率(SYMBOL_SWAP_LONG或SYMBOL | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | _SWAP_SHORT) | |
| SYMBOL_SWAP_SATURDAY | 周六到周日隔夜持仓的库存费计算比率 (SYMBOL_SWAP_LONG 或 SYMBOL_SWAP_SHORT) | 双精度 |
| SYMBOL_MARGIN_INITIAL | 原始预付款表示每一笔预付款开仓成交量的数量<br><br>SymbolInfoMarginRate() 函数根据订单类型和方向提供有关所收取预付款金额的数据 | 双精度 |
| SYMBOL_MARGIN_MAINTENANCE | 维持预付款，在交易品种中建立预付款数量，从一笔订单中获得预付款盈利，当 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 客户账户改变时，使用客户资产检测系统，如果维持预付款等于，使用原始预付款<br><br>SymbolInfoMarginRate() 函数根据订单类型和方向提供有关所收取预付款金额的数据 | |
| SYMBOL_SESSION_VOLUME | 当前交易的总交易量 | 双精度 |
| SYMBOL_SESSION_TURNOVER | 当前的总流通量 | 双精度 |
| SYMBOL_SESSION_INTEREST | 总持仓利息 | 双精度 |
| SYMBOL_SESSION_BUY_ORDERS_VOLUME | 买入订单的当前交易量 | 双精度 |
| SYMBOL_SESSION_SELL_ORDERS_VOLUME | 卖出订单的当前交易量 | 双精度 |
| SYMBOL_SESSION_OPEN | 当前持仓价格 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_SESSION_CLOSE | 当前平仓价格 | 双精度 |
| SYMBOL_SESSION_AW | 当前加权平均价格 | 双精度 |
| SYMBOL_SESSION_PRICE_SETTLEMENT | 当前结算价格 | 双精度 |
| SYMBOL_SESSION_PRICE_LIMIT_MIN | 当前最低价格 | 双精度 |
| SYMBOL_SESSION_PRICE_LIMIT_MAX | 当前最高价格 | 双精度 |
| SYMBOL_MARGIN_HEDGED | 锁仓持仓每手的合约大小或预付款值（一个交易品种的反向持仓）。锁仓持仓可以使用两种预付款计算方式。计算方式由交易商定义。<br><br>基本计算：<br>• 如果一个交易品种指定初始预付款(SYMBOL_MARGIN_INI | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | TIAL)，锁仓预付款被指定为一个绝对值（以货币计算）。<br><br>• 如果没有指定初始预付款（等于0），SYMBOL_MARGIN_HEDGED则等于合约大小，这将用来通过恰当公式根据金融工具类型计算预付款(SYMBOL_TRADE_CALC_M | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | ODE)。<br><br>计算最大持仓：<br>• SYMBOL_MARGIN_HEDGED值不被考虑。<br>• 计算交易品种所有出卖和入买持仓的交易量。<br>• 对于每个方向，都会计算加权均平开盘价和入金币货的加权平均转化率。<br>• 下一步，使用根据交易品种类型选定 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 的适当公式 (SYMBOL_TRADE_CALC_MODE) 计算卖出和买入持仓预付款。<br>• 最大值用作预付款。 | |
| SYMBOL_PRICE_CHANGE | 当前价格相对于前一交易日结束价格的变化（以百分比表示） | 双精度 |
| SYMBOL_PRICE_VOLATILITY | 价格波动率（以百分比表示） | 双精度 |
| SYMBOL_PRICE_THEORETICAL | 理论期权价格 | 双精度 |
| SYMBOL_PRICE_DELTA | 期权/warrant delta显示当基础资产价格变动1时，期权价 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 格变化的值 | |
| SYMBOL_PRICE_THETA | 期权/warrant theta显示期权价格每天因临时停止而损失的点数，即到期日临近时 | 双精度 |
| SYMBOL_PRICE_GAMMA | 期权/warrant gamma显示delta的变化率 - 期权溢价变化的快慢情况 | 双精度 |
| SYMBOL_PRICE_VEGA | 期权/warrant vega显示当波动率变化1%时，期权价格变化的点数 | 双精度 |
| SYMBOL_PRICE_RHO | 期权/warrant rho反映了理论期权价格对利率变化1%的敏感性 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_PRICE_OMEGA | 期权/warrant omega. 期权弹性显示了期权价格相对于基础资产价格变动的百分比 | 双精度 |
| SYMBOL_PRICE_SENSITIVITY | 期权/warrant sensitivity通过期权基础资产价格应该变化多少点，来显示期权价格应该变化一个点 | 双精度 |

函数 SymbolInfoString()

**ENUM_SYMBOL_INFO_STRING**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| SYMBOL_BASIS | 基础衍生资产 | 字符串 |
| SYMBOL_CATEGORY | 交易品种所属部分或类别的名称 | 字符串 |
| SYMBOL_COUNTRY | 交易品种所属的国家 | 字符串 |
| SYMBOL_SECTOR_NAME | 交易品种所属的经 | 字符串 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 济部门 | |
| SYMBOL_INDUSTRY_NAME | 交易品种所属的行业分支或行业 | 字符串 |
| SYMBOL_CURRENCY_BASE | 交易品种基础货币 | 字符串 |
| SYMBOL_CURRENCY_PROFIT | 利润货币 | 字符串 |
| SYMBOL_CURRENCY_MARGIN | 预付款货币 | 字符串 |
| SYMBOL_BANK | 当前报价支线 | 字符串 |
| SYMBOL_DESCRIPTION | 交易品种描述 | 字符串 |
| SYMBOL_EXCHANGE | 进行交易品种交易的交易所的名称 | 字符串 |
| SYMBOL_FORMULA | 用于自定义交易品种定价的公式。如果公式中使用的交易品种名称以数字开头或包含特殊字符(">" "", "." "-", "&", "#"), 那么应在交易品种名称外使用引号。<br>• 综合交易品种："@ | 字符串 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | ESU19"/EURCAD<br>• 日历点差："Si-9.13"-"Si-6.13"<br>• 欧元指数：34.3880 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 57266*pow(EURUSD,0.31555)*pow(EURGBP,0.30566)*pow(EU | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | RJPY,0.18911)*pow(EURCHF,0.11113)*pow(EURSEK,0.0788 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 5<br>) | |
| SYMBOL_ISIN | ISIN系统中交易品种的名称（国际证券识别号码）。国际证券识别号码是一个12位数字字母代码，是证券的唯一识别码。交易品种的属性是由交易服务器方面决定的 。 | 字符串 |
| SYMBOL_PAGE | 包含交易品种信息的网页地址。这个地址将显示为一个链接，用于在程序端中查看交易品种的属性 | 字符串 |
| SYMBOL_PATH | 交易品种树形通路 | 字符串 |

交易品种价格图表能够以卖价和买价为基础。为交易品种图表选择的价格也会影响程序端中柱形图的生成和展示。SYMBOL_CHART_MODE属性可能的数值描述在ENUM_SYMBOL_CHART_MODE

**ENUM_SYMBOL_CHART_MODE**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_CHART_MODE_BID | 柱形图以卖价为基础 |
| SYMBOL_CHART_MODE_LAST | 柱形图以最后价为基础 |

© 2000-2025, MetaQuotes Ltd.

待办订单中每个交易品种的几处终结点是能够制定的，一个标志匹配一种方式，使用 **OR** (|)逻辑操作符能够连接标志，例如，SYMBOL_EXPIRATION_GTC|SYMBOL_EXPIRATION_SPECIFIED，为了检测交易品种的某一方式是否被允许操作，结果可以使用**AND** (&)逻辑操作符来与方式标志相对比。

如果交易品种的SYMBOL_EXPIRATION_SPECIFIED标志可以指定，发送待办订单，可以为其指定某一时刻交易。

| 标识符 | 值 | 描述 |
| --- | --- | --- |
| SYMBOL_EXPIRATION_GTC | 1 | 该订单永久有效，直到它被明确取消 |
| SYMBOL_EXPIRATION_DAY | 2 | 订单在交易日当天有效 |
| SYMBOL_EXPIRATION_SPECIFIED | 4 | 订单中指定的到期时间 |
| SYMBOL_EXPIRATION_SPECIFIED_DAY | 8 | 订单中指定的到期日期 |

示例：

```
//+------------------------------------------------------------------+
//| 检测是否允许指定的到期模式                                       |
//+------------------------------------------------------------------+
bool IsExpirationTypeAllowed(string symbol,int exp_type)
  {
//--- 获得描述允许到期模式的属性值
   int expiration=(int)SymbolInfoInteger(symbol,SYMBOL_EXPIRATION_MODE);
//--- 如果exp_type模式被允许，返回true
   return((expiration&exp_type)==exp_type);
  }
```

如果SYMBOL_EXPIRATION_MODE属性被设置在SYMBOL_EXPIRATION_GTC (有效直至被取消)，挂单以及止损/止盈订单的到期时间应该使用ENUM_SYMBOL_ORDER_GTC_MODE枚举额外设置。

**ENUM_SYMBOL_ORDER_GTC_MODE**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_ORDERS_GTC | 挂单和止损/止盈水平永久有效，直至被明确取消 |
| SYMBOL_ORDERS_DAILY | 订单在一个交易日内有效。交易日结束，删除所有止损和止盈水平以及挂单。 |
| SYMBOL_ORDERS_DAILY_EXCLUDING_STOPS | 交易日变化时，只删除挂单，保留止损和止盈水平。 |

发送订单时，我们可以指定订单中设置的交易量的成交指令。表中指定了每个交易品种可能的基于交易量的订单执行选项。可以通过标识组合为每个交易品种设置多种模式。标识组合由逻

© 2000-2025, MetaQuotes Ltd.

辑OR (|)运算表示，例如SYMBOL_FILLING_FOK|SYMBOL_FILLING_IOC。要检查交易品种是否允许特定模式，对比逻辑AND (&)结果与模式标识 - 例如。

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| 全数执行或立即取消 | SYMBOL_FILLING_FOK | 1 | 订单只能按照指定的交易量执行。<br><br>如果金融产品的必要数额目前在市场上不能满足，则该订单将不会 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 被执行。所需交易量可以由几个可用报价组成。<br><br>当发送订单时，应该为该指令指定ORDER_FILL |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | NG_FOK成交类型。由交易服务器确定使用FOK订单的可能性。 |
| 立即或取消 | SYMBOL_FILLING_IOC | 2 | 交易者同意按照订单中指定的场内最 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 大交易量执行成交。如果不能完全成交，则将执行可用交易量的订单，并取消其余交易量。当发送订单时， |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 应该为该指令指定ORDER_FILLING_IOC成交类型。<br>由交易服务器确定使用IOC订单的可 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 能性。 |
| 被动 | SYMBOL_FILLING_BOC | 4 | BOC（预订或取消）策略假设订单只能在市场深度下单，不能立即执行。如果订单可以在下单后立即 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 执行，则取消订单。<br><br>事实上，只有在所下订单的价格不及当前市场价格时，才能指定此执行策略。BoC订 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 单用于实施被动交易，因此订单在下单时不会立即执行，也不会影响当前的流动性。仅支持限价单和止损限 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 价单，即SYMBOL_ORDER_MODE标识应包含SYMBOL_ORDER_LIMIT和/或S |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | YMBOL_ORDER_STOP_LIMIT值。 |
| 返回 | 无标识符 | | 如果部分成交，则市价单或限价单剩余交易量不会取消 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | ，会进一步处理。<br>当发送订单时，应该为该指令指定ORDER_FILLING_RETURN成交类 |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | 型。<br>在市价执行模式下不允许返回订单（市价执行 – SYMBOL_TRADE_EXECUTION_M |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | ID | 值 | 描述 |
| --- | --- | --- | --- |
| | | | ARKET)。 |

当使用OrderSend()函数发送交易请求时，所需的交易量执行指令可以在*type_filling*字段中设置，即在特殊的MqlTradeRequest结构中设置。可以使用ENUM_ORDER_TYPE_FILLING枚举值。如果没有指定成交类型，则在交易请求中自动设置ORDER_FILLING_RETURN。除了"市价执行"(SYMBOL_TRADE_EXECUTION_MARKET)外，将在任何执行模式中启用ORDER_FILLING_RETURN成交类型。

当**在当前时间**(有效时间)发送执行交易请求时，我们应该记住金融市场并不能保证某一交易品种可以期望的价格获得所需的全部交易量。因此，实时交易操作通过价格和交易量执行模式进行管理。模式或执行指令定义了价格改变或请求交易量不能在**此刻**完全满足的情况下的规则。

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| 执行模式<br><br>(请求执行) | 以之前从交易商接收的价格执行市价单。<br><br>订单发送之前，从经纪商处请求提供某个市价订单的价格。在收到价格后，按给定价格执 | SYMBOL_TRADE_EXECUTION_REQUEST |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| | 行的订单可以被确认或被回绝。 | |
| 现价执行<br><br>(Instant Execution) | 立即以指定价格执行市价单。<br><br>当发送要执行的交易请求时，平台自动将当前价格添加到订单中。<br>• 如果交易商接受该价格，则订单执行。<br>• 如果交 | SYMBOL_TRADE_EXECUTION_INSTANT |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| | 易商不接受请求价格，则发送"Requote"（重报价）—交易商订单可执行的价格。 | |
| 市价执行<br><br>(Market Execution) | 交易商需与易进额商不要交者行外量 | SYMBOL_TRADE_EXECUTION_MARKET |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| | 就可以决定订单执行价格。<br><br>以这种模式发送订单意味着事先同意以这个价格执行。 | |
| 交易所执行<br><br>(Exchange Execution) | 交易操作以当前市场报价执行。 | SYMBOL_TRADE_EXECUTION_EXCHANGE |

在发送当前执行时间的订单之前，为了正确设置ORDER_TYPE_FILLING值（交易量执行类型），您可以使用SymbolInfoInteger()函数（每个交易品种）来获取SYMBOL_FILLING_MODE属性值，将交易品种允许的交易量执行类型显示为标识组合。除了“市价执行”模式(SYMBOL_TRADE_EXECUTION_MARKET)外，将一直启用ORDER_FILLING_RETURN成交类型。

根据执行模式使用的成交类型可以如下表所示：

| 执行类型/成交指令类型 | Fill or Kill (FOK ORDER_FILLING_FOK) | Immediate or Cancel (IOC ORDER_FILLING_IOC) | Return (Return ORDER_FILLING_RETURN) |
| --- | --- | --- | --- |
| 现价执行<br><br>(SYMBOL_TRADE_EXECUTION_INSTANT) | + 无论交易品种如何设置） | + 无论交易品种如何设置） | + 始终） |
| 请求执行<br><br>SYMBOL_TRADE_EXECUTION_REQUEST | + 无论交易品种如何设置） | + 无论交易品种如何设置） | + 始终） |
| 市价执行 | + 在交易品种设置中设 | + 在交易品种设置中设 | - 无论交易品种如何设置， |

© 2000-2025, MetaQuotes Ltd.

| 执行类型/成交指令类型 | Fill or Kill (FOK ORDER_FILLING_FOK) | Immediate or Cancel (IOC ORDER_FILLING_IOC) | Return (Return ORDER_FILLING_RETURN) |
|---|---|---|---|
| SYMBOL_TRADE_EXECUTION_MARKET | 置） | 置） | 请禁用） |
| 交易所执行<br><br>SYMBOL_TRADE_EXECUTION_EXCHANGE | + 在交易品种设置中设置） | + 在交易品种设置中设置） | + 始终） |

挂单情况下，无论执行类型（SYMBOL_TRADE_EXEMODE）如何，都应该使用ORDER_FILLING_RETURN成交类型，因为这类订单不会在发送时执行。使用挂单时，交易者事先同意，当满足此订单的交易条件时，交易商将使用交易所支持的成交类型。

例如：

```
//+------------------------------------------------------------------+
//| 检查是否允许给定的成交模式                                       |
//+------------------------------------------------------------------+
bool IsFillingTypeAllowed(string symbol,int fill_type)
  {
//--- 获取描述成交模式的属性值
   int filling=(int)SymbolInfoInteger(symbol,SYMBOL_FILLING_MODE);
//--- 如果允许fill_type模式，则返回'true'
   return((filling&fill_type)==fill_type);
  }
```

当使用OrderSend()函数发送交易请求时，来自ENUM_ORDER_TYPE枚举的订单类型应该被指定一些操作。不是所有的订单类型都可以允许指定交易品种。SYMBOL_ORDER_MODE属性描述了允许的订单类型的标识。

| 标识符 | 值 | 描述 |
|---|---|---|
| SYMBOL_ORDER_MARKET | 1 | 允许市价单（买入和卖出） |
| SYMBOL_ORDER_LIMIT | 2 | 允许限价单（买入限价和卖出限价） |
| SYMBOL_ORDER_STOP | 4 | 允许止损订单（买入止损和卖出止损） |
| SYMBOL_ORDER_STOP_LIMIT | 8 | 允许止损限价订单（买入止损限价和卖出止损限价） |
| SYMBOL_ORDER_SL | 16/ | 允许止损 |
| SYMBOL_ORDER_TP | 32 | 允许获利 |
| SYMBOL_ORDER_CLOSEBY | 64 | Close By操作权限，例如，通过相同工具但方向相反的另一个持仓来平仓。该属性是为锁仓账户系统的账户而设置 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 值 | 描述 |
| --- | --- | --- |
| | | (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING) |

例如：

```
//+------------------------------------------------------------------+
//| 该函数打印出交易品种允许的订单类型                                |
//+------------------------------------------------------------------+
void Check_SYMBOL_ORDER_MODE(string symbol)
  {
//--- 接收描述允许订单类型的属性的值
   int symbol_order_mode=(int)SymbolInfoInteger(symbol,SYMBOL_ORDER_MODE);
//--- 检查市价单（市场执行）
   if((SYMBOL_ORDER_MARKET&symbol_order_mode)==SYMBOL_ORDER_MARKET)
      Print(symbol+": Market orders are allowed (Buy and Sell)");
//--- 检查限价单
   if((SYMBOL_ORDER_LIMIT&symbol_order_mode)==SYMBOL_ORDER_LIMIT)
      Print(symbol+": Buy Limit and Sell Limit orders are allowed");
//--- 检查止损单
   if((SYMBOL_ORDER_STOP&symbol_order_mode)==SYMBOL_ORDER_STOP)
      Print(symbol+": Buy Stop and Sell Stop orders are allowed");
//--- 检查止损限价订单
   if((SYMBOL_ORDER_STOP_LIMIT&symbol_order_mode)==SYMBOL_ORDER_STOP_LIMIT)
      Print(symbol+": Buy Stop Limit and Sell Stop Limit orders are allowed");
//--- 检查是否允许下止损订单
   if((SYMBOL_ORDER_SL&symbol_order_mode)==SYMBOL_ORDER_SL)
      Print(symbol+": Stop Loss orders are allowed");
//--- 检查是否允许下获利订单
   if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_TP)
      Print(symbol+": Take Profit orders are allowed");
//--- 检查是否允许反向平仓
   if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_CLOSEBY)
      Print(symbol+": Close by allowed");
//---
  }
```

为计算交易品种的预付款规定额，使用The ENUM_SYMBOL_CALC_MODE项目获取信息。

**ENUM_SYMBOL_CALC_MODE**

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| SYMBOL_CALC_MODE_FOREX | 外汇模式—计算盈利和预付款 | 预付款： Lots * Contract_Size / Leverage Contract_Size / Leverage * Margin_Rate<br><br>利润：(close_price - open_price) * Contract_Size * Lots |
| SYMBOL_CALC_MODE_FOREX_NO_LEVERAGE | 外汇无杠 | 预付款：Lots * Contract_Size * Margin_Rate |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 杆模式 — 不考虑杠杆作用的情况下来计算外汇交易品种的利润和预付款 | 利润 : (close_price - open_price) * Contract_Size * Lots |
| SYMBOL_CALC_MODE_FUTURES | 期货模式—计算预付款和盈利 | 预付款：Lots * InitialMargin * Margin_Rate<br><br>利润：(close_price - open_price) * TickPrice / TickSize * Lots |
| SYMBOL_CALC_MODE_CFD | CFD 模式—计算预付款和盈利 | 预付款：Lots * ContractSize * MarketPrice * Margin_Rate<br><br>盈利：(close_price - open_price) * Contract_Size * Lots |
| SYMBOL_CALC_MODE_CFDINDEX | CFD 指数模式—通过指数计算预付款和盈利 | 预付款：(Lots * ContractSize * MarketPrice) * TickPrice / TickSize * Margin_Rate<br><br>盈利：(close_price - open_price) * Contract_Size * Lots |
| SYMBOL_CALC_MODE_CFDLEVERAGE | CFD 杠杆模式—通过杠杆贸 | 预付款：(Lots * ContractSize * MarketPrice) / Leverage * Margin_Rate<br><br>盈利：(close_price - open_price) * Contract_Size * Lots |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 计算预付款和盈利 | |
| SYMBOL_CALC_MODE_EXCH_STOCKS | 交换模式 - 计算股票交易中，交易证券的预付款和赢利 | 预付款：Lots * ContractSize * LastPrice * Margin_Rate<br><br>盈利：(close_price - open_price) * Contract_Size * Lots |
| SYMBOL_CALC_MODE_EXCH_FUTURES | 期货模式 - 计算股票交易中，交易期货合约的预付款和赢利 | 预付款：Lots * InitiaMargin or Lots * MaintenanceMargin * Margin_Rate<br><br>盈利：(close_price - open_price) * Lots*TickPrice / TickSize |
| SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS | 期货期权模式 - 计算期权交易中，交易期货合约的预付款和赢 | 预付款：Lots * InitiaMargin or Lots * MaintenanceMargin * Margin_Rate<br><br>盈利：(close_price - open_price) * Lots * TickPrice / TickSize |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 利。根据以下规则，预付款可能会因MarginDiscount偏差的数额而减少：1. 如果买入持仓的价格（买入订单）少于预估价格，MarginDiscount = Lots*((PriceSettle-PriceOrder)*TickPrice/TickSize)2. 如果卖出持 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 仓的价格（卖出订单）超过预估价格，MarginDiscount = Lots*((PriceOrder-PriceSettle)*TickPrice/TickSize)<br>在这里：<br>○ PriceSettle - 前一交易日的预估价 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | ○ 格（清算）；PriceOrder - 订单（请求）中设置的加权平均持仓价格或持仓价格；<br>○ TickPri | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | ce - 订单价（成本价格的点变化）<br>○ TickSize - 订单大小（最低价格变化的步骤） | |
| SYMBOL_CALC_MODE_EXCH_BONDS | 交易所债券模式 - 为在票交易 | 预付款：Lots * ContractSize * FaceValue * open_price * /100<br><br>盈利：Lots * close_price * FaceValue * Contract_Size + AccruedInterest * Lots * ContractSize |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 所交易债券计算预付款和盈利 | |
| SYMBOL_CALC_MODE_EXCH_STOCKS_MOEX | 交易所股票模式(MOEX) - 为在MOEX交易证券计算预付款和盈利 | 预付款：Lots*ContractSize*LastPrice*Margin_Rate<br><br>盈利：(close_price-open_price)*Contract_Size*Lots |
| SYMBOL_CALC_MODE_EXCH_BONDS_MOEX | 交易所债券模式(MOEX) - 为在MOEX交易债券计算预付款和盈利 | 预付款：Lots*ContractSize*FaceValue*open_price*/100<br><br>盈利：Lots*close_price*FaceValue*Contract_Size+AccruedInterest*Lots*ContractSize |
| SYMBOL_CALC_MODE_SERV_COLLATERAL | 担保模式 - 在交易账户中使用交易品种作 | 预付款：no<br>利润：no<br><br>市值：Lots*ContractSize*MarketPrice*LiquidityRate |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | 非易仓市根交，前，约小流比进计。值括资，加到。种易种持提了用付的，用可易具仓额预款为交资产。持的值据易量，当市价，合大和动率行算。价包在产中，被入净值。这交品的仓高可预款数额，并作交工持的外付 | |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 公式 |
| --- | --- | --- |
| | （担保）。 | |

交易品种有许多种交易方式，某一交易品种的交易方式信息反映在ENUM_SYMBOL_TRADE_MODE枚举值中。

**ENUM_SYMBOL_TRADE_MODE**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_TRADE_MODE_DISABLED | 交易品种禁止交易 |
| SYMBOL_TRADE_MODE_LONGONLY | 仅允许买入持仓 |
| SYMBOL_TRADE_MODE_SHORTONLY | 仅允许卖出持仓 |
| SYMBOL_TRADE_MODE_CLOSEONLY | 仅允许平仓操作 |
| SYMBOL_TRADE_MODE_FULL | 没有交易限制 |

ENUM_SYMBOL_TRADE_EXECUTION枚举中定义每一个交易品种的可能的订单执行方式

**ENUM_SYMBOL_TRADE_EXECUTION**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_TRADE_EXECUTION_REQUEST | 请求执行 |
| SYMBOL_TRADE_EXECUTION_INSTANT | 立即执行 |
| SYMBOL_TRADE_EXECUTION_MARKET | 市价执行 |
| SYMBOL_TRADE_EXECUTION_EXCHANGE | 交换执行 |

在枚举ENUM_SYMBOL_SWAP_MODE中指定的仓位转移利息计算方式。利息计算方式决定了 SYMBOL_SWAP_LONG 和 SYMBOL_SWAP_SHORT 参数的测量单位。例如，如果客户入金货币收取利息，那么那些参数的值则指定为客户入金货币的金额。

**ENUM_SYMBOL_SWAP_MODE**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_SWAP_MODE_DISABLED | 禁用利息（无利息） |
| SYMBOL_SWAP_MODE_POINTS | 以点数收取利息 |
| SYMBOL_SWAP_MODE_CURRENCY_SYMBOL | 以交易品种基础货币的钱数收取利息 |
| SYMBOL_SWAP_MODE_CURRENCY_MARGIN | 以交易品种预付款货币的钱数收取利息 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT | 以客户入金货币的钱数收取利息 |
| SYMBOL_SWAP_MODE_CURRENCY_PROFIT | 库存费以利润计算货币计价 |
| SYMBOL_SWAP_MODE_INTEREST_CURRENT | 利息作为指定的年利息，从计算利息的工具价格收取（标准银行年是360天） |
| SYMBOL_SWAP_MODE_INTEREST_OPEN | 利息作为指定的年利息，从持仓的开盘价格收取（标准银行年是360天） |
| SYMBOL_SWAP_MODE_REOPEN_CURRENT | 重新持仓收取利息。交易日结束，平仓。第二天以收盘价 +/- 指定点数（参数SYMBOL_SWAP_LONG和SYMBOL_SWAP_SHORT）重新持仓 |
| SYMBOL_SWAP_MODE_REOPEN_BID | 重新持仓收取利息。交易日结束，平仓。第二天以当前卖价 +/- 指定点数（参数SYMBOL_SWAP_LONG和SYMBOL_SWAP_SHORT）重新持仓 |

使用ENUM_DAY_OF_WEEK枚举值描述工作日

**ENUM_DAY_OF_WEEK**

| 标识符 | 描述 |
| --- | --- |
| SUNDAY | 星期日 |
| MONDAY | 星期一 |
| TUESDAY | 星期二 |
| WEDNESDAY | 星期三 |
| THURSDAY | 星期四 |
| FRIDAY | 星期五 |
| SATURDAY | 星期六 |

期权是一份合约，给予的是权限，而非责任，为了在指定价格或在特定日期前买入或出售基础资产（商品，股票，期货等等）。以下枚举描述了期权的属性，包括期权类型和由其产生的权限。

**ENUM_SYMBOL_OPTION_RIGHT**

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_OPTION_RIGHT_CALL | 看涨期权就是提供给您在指定价格购买资产的权限 |
| SYMBOL_OPTION_RIGHT_PUT | 看跌期权就是提供给您在特定价格出售资产的权限 |

© 2000-2025, MetaQuotes Ltd.

ENUM_SYMBOL_OPTION_MODE

| 标识符 | 描述 |
| --- | --- |
| SYMBOL_OPTION_MODE_EUROPEAN | 欧式期权可能只能在特定日期行使 (到期，执行日期，交付日期) |
| SYMBOL_OPTION_MODE_AMERICAN | 美式期权可以在任何交易日或到期之前行使。在指定周期买家可以行使期权 |

交易品种按照经济部门进行分类。经济部门是经济活动的一部分，具有特定的特征、经济目标、功能和行为，可以使该部门与经济的其他部分分开。ENUM_SYMBOL_SECTOR列出交易品种可能所属的经济部门。

ENUM_SYMBOL_SECTOR

| ID | 描述 |
| --- | --- |
| SECTOR_UNDEFINED | 未定义 |
| SECTOR_BASIC_MATERIALS | 基本材料 |
| SECTOR_COMMUNICATION_SERVICES | 通讯服务 |
| SECTOR_CONSUMER_CYCLICAL | 消费类周期性 |
| SECTOR_CONSUMER_DEFENSIVE | 消费防御 |
| SECTOR_CURRENCY | 货币 |
| SECTOR_CURRENCY_CRYPTO | 数字加密货币 |
| SECTOR_ENERGY | 能源 |
| SECTOR_FINANCIAL | 财务 |
| SECTOR_HEALTHCARE | 医疗保健 |
| SECTOR_INDUSTRIALS | 工业 |
| SECTOR_REAL_ESTATE | 房地产 |
| SECTOR_TECHNOLOGY | 技术 |
| SECTOR_UTILITIES | 公用事业 |

可以将每个交易品种分配到特定类型的行业或经济分支。工业是经济的一个分支，生产一组紧密相关的原材料、商品或服务。ENUM_SYMBOL_INDUSTRY列出交易品种可能所属的行业。

ENUM_SYMBOL_INDUSTRY

| ID | 值 | 描述 |
| --- | --- | --- |
| INDUSTRY_UNDEFINED | 0 | 未定义 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
|---|---|---|
| 基本材料 | | |
| INDUSTRY_AGRICULTURAL_INPUTS | 1 | 农业投入 |
| INDUSTRY_ALUMINIUM | 2 | 铝 |
| INDUSTRY_BUILDING_MATERIALS | 3 | 建筑材料 |
| INDUSTRY_CHEMICALS | 4 | 化学制品 |
| INDUSTRY_COKING_COAL | 5 | 焦煤 |
| INDUSTRY_COPPER | 6 | 铜 |
| INDUSTRY_GOLD | 7 | 黄金 |
| INDUSTRY_LUMBER_WOOD | 8 | 木材和木材生产 |
| INDUSTRY_INDUSTRIAL_METALS | 9 | 其他工业金属和矿业 |
| INDUSTRY_PRECIOUS_METALS | 10 | 其他贵金属和矿业 |
| INDUSTRY_PAPER | 11 | 造纸及纸制品 |
| INDUSTRY_SILVER | 12 | 白银 |
| INDUSTRY_SPECIALTY_CHEMICALS | 13 | 特种化学品 |
| INDUSTRY_STEEL | 14 | 钢铁 |
| 通讯服务 | | |
| INDUSTRY_ADVERTISING | 51 | 广告代理 |
| INDUSTRY_BROADCASTING | 52 | 广播 |
| INDUSTRY_GAMING_MULTIMEDIA | 53 | 电子游戏和多媒体 |
| INDUSTRY_ENTERTAINMENT | 54 | 娱乐 |
| INDUSTRY_INTERNET_CONTENT | 55 | 互联网内容和信息 |
| INDUSTRY_PUBLISHING | 56 | 出版 |
| INDUSTRY_TELECOM | 57 | 电信服务 |
| 消费类周期性 | | |
| INDUSTRY_APPAREL_MANUFACTURING | 101 | 服装制造业 |
| INDUSTRY_APPAREL_RETAIL | 102 | 服装零售 |
| INDUSTRY_AUTO_MANUFACTURERS | 103 | 汽车制造商 |
| INDUSTRY_AUTO_PARTS | 104 | 汽车零部件 |
| INDUSTRY_AUTO_DEALERSHIP | 105 | 汽车及卡车经销商 |
| INDUSTRY_DEPARTMENT_STORES | 106 | 百货商店 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
| --- | --- | --- |
| INDUSTRY_FOOTWEAR_ACCESSORIES | 107 | 鞋类及配件 |
| INDUSTRY_FURNISHINGS | 108 | 家具、固定资产和家用电器 |
| INDUSTRY_GAMBLING | 109 | 赌博 |
| INDUSTRY_HOME_IMPROV_RETAIL | 110 | 家居装修零售 |
| INDUSTRY_INTERNET_RETAIL | 111 | 互联网零售 |
| INDUSTRY_LEISURE | 112 | 休闲娱乐 |
| INDUSTRY_LODGING | 113 | 住宿 |
| INDUSTRY_LUXURY_GOODS | 114 | 奢侈品 |
| INDUSTRY_PACKAGING_CONTAINERS | 115 | 包装和容器 |
| INDUSTRY_PERSONAL_SERVICES | 116 | 个人服务 |
| INDUSTRY_RECREATIONAL_VEHICLES | 117 | 旅游车 |
| INDUSTRY_RESIDENT_CONSTRUCTION | 118 | 住宅建筑 |
| INDUSTRY_RESORTS_CASINOS | 119 | 度假村和赌场 |
| INDUSTRY_RESTAURANTS | 120 | 餐厅 |
| INDUSTRY_SPECIALTY_RETAIL | 121 | 特色零售业 |
| INDUSTRY_TEXTILE_MANUFACTURING | 122 | 纺织制造业 |
| INDUSTRY_TRAVEL_SERVICES | 123 | 旅游服务业 |
| 消费防御 | | |
| INDUSTRY_BEVERAGES_BREWERS | 151 | 饮料－啤酒 |
| INDUSTRY_BEVERAGES_NON_ALCO | 152 | 饮料－非酒精饮料 |
| INDUSTRY_BEVERAGES_WINERIES | 153 | 饮料－葡萄酒厂和酿酒厂 |
| INDUSTRY_CONFECTIONERS | 154 | 糖果店 |
| INDUSTRY_DISCOUNT_STORES | 155 | 折扣店 |
| INDUSTRY_EDUCATION_TRAINIG | 156 | 教育和培训服务 |
| INDUSTRY_FARM_PRODUCTS | 157 | 农产品 |
| INDUSTRY_FOOD_DISTRIBUTION | 158 | 食品分配 |
| INDUSTRY_GROCERY_STORES | 159 | 杂货店 |
| INDUSTRY_HOUSEHOLD_PRODUCTS | 160 | 家庭及个人用品 |
| INDUSTRY_PACKAGED_FOODS | 161 | 包装食品 |
| INDUSTRY_TOBACCO | 162 | 烟草 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
| --- | --- | --- |
| 能源 | | |
| INDUSTRY_OIL_GAS_DRILLING | 201 | 油气钻井 |
| INDUSTRY_OIL_GAS_EP | 202 | 石油和天然气的开采和加工 |
| INDUSTRY_OIL_GAS_EQUIPMENT | 203 | 油气设备与服务 |
| INDUSTRY_OIL_GAS_INTEGRATED | 204 | 油气一体化 |
| INDUSTRY_OIL_GAS_MIDSTREAM | 205 | 油气中游 |
| INDUSTRY_OIL_GAS_REFINING | 206 | 油气炼制与营销 |
| INDUSTRY_THERMAL_COAL | 207 | 热能煤 |
| INDUSTRY_URANIUM | 208 | 铀 |
| 财务 | | |
| INDUSTRY_EXCHANGE_TRADED_FUND | 251 | 交易所买卖基金 |
| INDUSTRY_ASSETS_MANAGEMENT | 252 | 资产管理 |
| INDUSTRY_BANKS_DIVERSIFIED | 253 | 银行 - 全国银行 |
| INDUSTRY_BANKS_REGIONAL | 254 | 银行 - 地方银行 |
| INDUSTRY_CAPITAL_MARKETS | 255 | 资本市场 |
| INDUSTRY_CLOSE_END_FUND_DEBT | 256 | 封闭式基金 - 债务 |
| INDUSTRY_CLOSE_END_FUND_EQUITY | 257 | 封闭式基金 - 股票类 |
| INDUSTRY_CLOSE_END_FUND_FOREIGN | 258 | 封闭式基金 - 外国 |
| INDUSTRY_CREDIT_SERVICES | 259 | 信贷服务 |
| INDUSTRY_FINANCIAL_CONGLOMERATE | 260 | 金融集团 |
| INDUSTRY_FINANCIAL_DATA_EXCHANGE | 261 | 财务数据和证券交易所 |
| INDUSTRY_INSURANCE_BROKERS | 262 | 保险经纪人 |
| INDUSTRY_INSURANCE_DIVERSIFIED | 263 | 保险 - 综合险 |
| INDUSTRY_INSURANCE_LIFE | 264 | 保险 - 寿险 |
| INDUSTRY_INSURANCE_PROPERTY | 265 | 保险 - 财产险和意外险 |
| INDUSTRY_INSURANCE_REINSURANCE | 266 | 保险 - 再保险 |
| INDUSTRY_INSURANCE_SPECIALTY | 267 | 保险 - 专业 |
| INDUSTRY_MORTGAGE_FINANCE | 268 | 按揭贷款 |
| INDUSTRY_SHELL_COMPANIES | 269 | 空壳公司 |
| 医疗保健 | | |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
| --- | --- | --- |
| INDUSTRY_BIOTECHNOLOGY | 301 | 生物技术 |
| INDUSTRY_DIAGNOSTICS_RESEARCH | 302 | 诊断与研究 |
| INDUSTRY_DRUGS_MANUFACTURERS | 303 | 药品制造商 – 普通 |
| INDUSTRY_DRUGS_MANUFACTURERS_SPEC | 304 | 药品制造商 – 专业和通用 |
| INDUSTRY_HEALTHCARE_PLANS | 305 | 医疗保健计划 |
| INDUSTRY_HEALTH_INFORMATION | 306 | 健康信息服务 |
| INDUSTRY_MEDICAL_FACILITIES | 307 | 医疗保健设施 |
| INDUSTRY_MEDICAL_DEVICES | 308 | 医疗设备 |
| INDUSTRY_MEDICAL_DISTRIBUTION | 309 | 医疗分配 |
| INDUSTRY_MEDICAL_INSTRUMENTS | 310 | 医疗器械和用品 |
| INDUSTRY_PHARM_RETAILERS | 311 | 药品零售商 |
| 工业 | | |
| INDUSTRY_AEROSPACE_DEFENSE | 351 | 航空航天与国防 |
| INDUSTRY_AIRLINES | 352 | 航空公司 |
| INDUSTRY_AIRPORTS_SERVICES | 353 | 机场和航空服务 |
| INDUSTRY_BUILDING_PRODUCTS | 354 | 建筑产品及设备 |
| INDUSTRY_BUSINESS_EQUIPMENT | 355 | 商业设备及用品 |
| INDUSTRY_CONGLOMERATES | 356 | 企业集团 |
| INDUSTRY_CONSULTING_SERVICES | 357 | 咨询服务 |
| INDUSTRY_ELECTRICAL_EQUIPMENT | 358 | 电气设备及零件 |
| INDUSTRY_ENGINEERING_CONSTRUCTION | 359 | 工程建设 |
| INDUSTRY_FARM_HEAVY_MACHINERY | 360 | 农用重型工程机械 |
| INDUSTRY_INDUSTRIAL_DISTRIBUTION | 361 | 产业分布 |
| INDUSTRY_INFRASTRUCTURE_OPERATIONS | 362 | 基础设施运营 |
| INDUSTRY_FREIGHT_LOGISTICS | 363 | 综合货运与物流 |
| INDUSTRY_MARINE_SHIPPING | 364 | 海运 |
| INDUSTRY_METAL_FABRICATION | 365 | 金属加工 |
| INDUSTRY_POLLUTION_CONTROL | 366 | 污染与处理控制 |
| INDUSTRY_RAILROADS | 367 | 铁路 |
| INDUSTRY_RENTAL_LEASING | 368 | 租赁服务 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
| --- | --- | --- |
| INDUSTRY_SECURITY_PROTECTION | 369 | 安全防护服务 |
| INDUSTRY_SPEALITY_BUSINESS_SERVICES | 370 | 特色商务服务 |
| INDUSTRY_SPEALITY_MACHINERY | 371 | 特种工业机械 |
| INDUSTRY_STUFFING_EMPLOYMENT | 372 | 原料和就业服务 |
| INDUSTRY_TOOLS_ACCESSORIES | 373 | 工具和配件 |
| INDUSTRY_TRUCKING | 374 | 货车运输 |
| INDUSTRY_WASTE_MANAGEMENT | 375 | 垃圾管理 |
| 房地产 | | |
| INDUSTRY_REAL_ESTATE_DEVELOPMENT | 401 | 房地产 - 开发 |
| INDUSTRY_REAL_ESTATE_DIVERSIFIED | 402 | 房地产 - 多元化 |
| INDUSTRY_REAL_ESTATE_SERVICES | 403 | 房地产服务 |
| INDUSTRY_REIT_DIVERSIFIED | 404 | REIT（房地产投资信托）- 多元化 |
| INDUSTRY_REIT_HEALTCARE | 405 | REIT（房地产投资信托）- 医疗保健 |
| INDUSTRY_REIT_HOTEL_MOTEL | 406 | REIT（房地产投资信托）- 酒店和汽车旅馆 |
| INDUSTRY_REIT_INDUSTRIAL | 407 | REIT（房地产投资信托）- 工业 |
| INDUSTRY_REIT_MORTAGE | 408 | REIT（房地产投资信托）- 抵押 |
| INDUSTRY_REIT_OFFICE | 409 | REIT（房地产投资信托）- 办公室 |
| INDUSTRY_REIT_RESIDENTAL | 410 | REIT（房地产投资信托）- 住宅 |
| INDUSTRY_REIT_RETAIL | 411 | REIT（房地产投资信托）- 零售 |
| INDUSTRY_REIT_SPECIALITY | 412 | REIT（房地产投资信托）- 专业 |
| 技术 | | |
| INDUSTRY_COMMUNICATION_EQUIPMENT | 451 | 通讯设备 |
| INDUSTRY_COMPUTER_HARDWARE | 452 | 计算机硬件 |
| INDUSTRY_CONSUMER_ELECTRONICS | 453 | 家用电子产品 |
| INDUSTRY_ELECTRONIC_COMPONENTS | 454 | 电子元器件 |
| INDUSTRY_ELECTRONIC_DISTRIBUTION | 455 | 电子与计算机分销 |
| INDUSTRY_IT_SERVICES | 456 | 信息技术服务 |
| INDUSTRY_SCIENTIFIC_INSTRUMENTS | 457 | 科技仪器 |
| INDUSTRY_SEMICONDUCTOR_EQUIPMENT | 458 | 半导体设备与材料 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 描述 |
| --- | --- | --- |
| INDUSTRY_SEMICONDUCTORS | 459 | 半导体 |
| INDUSTRY_SOFTWARE_APPLICATION | 460 | 软件 - 应用 |
| INDUSTRY_SOFTWARE_INFRASTRUCTURE | 461 | 软件 - 基础架构 |
| INDUSTRY_SOLAR | 462 | 太阳能 |
| 公用事业 | | |
| INDUSTRY_UTILITIES_DIVERSIFIED | 501 | 公用事业 - 多元化 |
| INDUSTRY_UTILITIES_POWERPRODUCERS | 502 | 公用事业 - 独立电力生产商 |
| INDUSTRY_UTILITIES_RENEWABLE | 503 | 公用事业 - 可再生能源 |
| INDUSTRY_UTILITIES_REGULATED_ELECTRIC | 504 | 公用事业 - 调节电 |
| INDUSTRY_UTILITIES_REGULATED_GAS | 505 | 公用事业 - 调节气 |
| INDUSTRY_UTILITIES_REGULATED_WATER | 506 | 公用事业 - 调节水 |
| INDUSTRY_UTILITIES_FIRST | 501 | 启动公用事业服务类型枚举。对应于 INDUSTRY_UTILITIES_DIVERSIFIED。 |
| INDUSTRY_UTILITIES_LAST | 506 | 公用事业服务类型枚举结束。对应于 INDUSTRY_UTILITIES_REGULATED_WATER。 |

© 2000-2025, MetaQuotes Ltd.

# 账户属性

使用 AccountInfoInteger(), AccountInfoDouble() 和 AccountInfoString() 函数获得活期账户信息，该函数参数值可用接收类似于ENUM_ACCOUNT_INFO项目的值。

使用 AccountInfoInteger() 函数

**ENUM_ACCOUNT_INFO_INTEGER**

| 标识符 | 描述 | 类型 |
|---|---|---|
| ACCOUNT_LOGIN | 账户数量 | long |
| ACCOUNT_TRADE_MODE | 账户交易方式 | ENUM_ACCOUNT_TRADE_MODE |
| ACCOUNT_LEVERAGE | 账户杠杆 | long |
| ACCOUNT_LIMIT_ORDERS | 活跃待办订单最大允许量 | int |
| ACCOUNT_MARGIN_SO_MODE | 建立最少使用保证金方式 | ENUM_ACCOUNT_STOPOUT_MODE |
| ACCOUNT_TRADE_ALLOWED | 允许活期账户交易 | bool |
| ACCOUNT_TRADE_EXPERT | 允许EA交易 | bool |
| ACCOUNT_MARGIN_MODE | 预付款计算模式 | ENUM_ACCOUNT_MARGIN_MODE |
| ACCOUNT_CURRENCY_DIGITS | 账户货币的小数位数，这是正确展示交易结果所必需的 | int |
| ACCOUNT_FIFO_CLOSE | 表示只能通过FIFO规则平仓 | bool |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 的一种指示。如果该属性值设为true，那么每个交易品种都将按照持仓的相同顺序进行平仓，从最早持仓的开始。如果试图以不同的顺序平仓，那么交易者将收到一个对应的错误提示。<br><br>对于采用非锁仓持仓账户模式的账户(ACCOUNT_MARGIN_MODE!=ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，该属性值始终为false。 | |

© 2000-2025, MetaQuotes Ltd.

函数 AccountInfoDouble()

ENUM_ACCOUNT_INFO_DOUBLE

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| ACCOUNT_BALANCE | 存入货币时账户结余 | 双精度 |
| ACCOUNT_CREDIT | 存入货币时账户亏空 | 双精度 |
| ACCOUNT_PROFIT | 存入货币时账户当前利润 | 双精度 |
| ACCOUNT_EQUITY | 存入货币时账户权益 | 双精度 |
| ACCOUNT_MARGIN | 存入货币时账户保证金使用 | 双精度 |
| ACCOUNT_MARGIN_FREE | 存入货币时账户可用保证金 | 双精度 |
| ACCOUNT_MARGIN_LEVEL | 账户保证金水平仪百分比形式制定 | 双精度 |
| ACCOUNT_MARGIN_SO_CALL | 保证金调用水平，依据建立的ACCOUNT_MARGIN_SO_MODE，以百分比形式或存入货币时期表示 | 双精度 |
| ACCOUNT_MARGIN_SO_SO | 保证金停用水平，依据建立的ACCOUNT_MARGIN_SO_MODE，以百分比 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 形式或存入货币时期表示 | |
| ACCOUNT_MARGIN_INITIAL | 初始预付款。账户上保留的金额需要涵盖所有挂单的预付款 | 双精度 |
| ACCOUNT_MARGIN_MAINTENANCE | 维持预付款。账户上保留的最小净值需要涵盖所有持仓的最小金额 | 双精度 |
| ACCOUNT_ASSETS | 账户的活动资本 | 双精度 |
| ACCOUNT_LIABILITIES | 账户的流动负债 | 双精度 |
| ACCOUNT_COMMISSION_BLOCKED | 账户的当前锁定手续费金额 | 双精度 |

函数 AccountInfoString()

**ENUM_ACCOUNT_INFO_STRING**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| ACCOUNT_NAME | 用户名 | 字符串 |
| ACCOUNT_SERVER | 交易服务器名称 | 字符串 |
| ACCOUNT_CURRENCY | 账户货币 | 字符串 |
| ACCOUNT_COMPANY | 提供账户的公司名称 | 字符串 |

有几种账户类型，可以在交易服务器上打开。MQL5程序运行的账户类型可以使用ENUM_ACCOUNT_TRADE_MODE枚举找出来。

**ENUM_ACCOUNT_TRADE_MODE**

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| ACCOUNT_TRADE_MODE_DEMO | 样本账户 |
| ACCOUNT_TRADE_MODE_CONTEST | 竞争账户 |
| ACCOUNT_TRADE_MODE_REAL | 真实账户 |

如果净值不足以保持持仓，止损离场的情况，例如强制平仓发生。出现止损离场的最小预付款水平能够以百分比或货币条款来设置。若要找到账户设置的模式，请使用ENUM_ACCOUNT_STOPOUT_MODE枚举。

**ENUM_ACCOUNT_STOPOUT_MODE**

| 标识符 | 描述 |
| --- | --- |
| ACCOUNT_STOPOUT_MODE_PERCENT | 账户以半分比方式截止 |
| ACCOUNT_STOPOUT_MODE_MONEY | 账户以货币方式截止 |

**ENUM_ACCOUNT_MARGIN_MODE**

| 标识符 | 描述 |
| --- | --- |
| ACCOUNT_MARGIN_MODE_RETAIL_NETTING | 用于场外交易市场“单边持仓”模式下的持仓说明（一个交易品种只存在一个持仓）。预付款计算基于交易品种的类型(SYMBOL_TRADE_CALC_MODE)。 |
| ACCOUNT_MARGIN_MODE_EXCHANGE | 用于交易所市场。预付款计算基于交易品种设置中指定的折扣。折扣由交易商设定，但不能低于交易所设定的值。 |
| ACCOUNT_MARGIN_MODE_RETAIL_HEDGING | 用于交易所市场，在这里个体持仓成为可能（锁仓，一个交易品种可以存在多个持仓）。预付款计算基于考虑锁仓预付款(SYMBOL_MARGIN_HEDGED)的交易品种类型(SYMBOL_TRADE_CALC_MODE)。 |

输出简单账户信息的脚本示例。

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 公司名称
   string company=AccountInfoString(ACCOUNT_COMPANY);
//--- 客户名称
   string name=AccountInfoString(ACCOUNT_NAME);
//--- 账号
   long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- 服务器名称
   string server=AccountInfoString(ACCOUNT_SERVER);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 账户货币
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- 模拟，竞赛或真实账户
   ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(AC
//--- 现在将枚举的值转换为一个可理解的形式
   string trade_mode;
   switch(account_type)
     {
      case  ACCOUNT_TRADE_MODE_DEMO:
         trade_mode="demo";
         break;
      case  ACCOUNT_TRADE_MODE_CONTEST:
         trade_mode="contest";
         break;
      default:
         trade_mode="real";
         break;
     }
//--- 以百分比或货币形式设置止损离场
   ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteg
//--- 获取当出现预付款调用和止损离场时的水平的值
   double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
   double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- 显示账户信息简介
   PrintFormat("The account of the client '%s' #%d %s opened in '%s' on the server '%s
               name,login,trade_mode,company,server);
   PrintFormat("Account currency - %s, MarginCall and StopOut levels are set in %s",
               currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"percentage":" m
   PrintFormat("MarginCall=%G, StopOut=%G",margin_call,stop_out);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 测试统计

测试结束之后，会计算不同的交易结果统计的参数。通过指定来自ENUM_STATISTICS 枚举的参数ID，可以使用 TesterStatistics() 函数来获得参数值。

虽然两种类型的参数（int和double）都用于计算统计，但是函数会以double格式返回全部的值。double型的所有统计值默认以入金货币表示，除非另有说明。

**ENUM_STATISTICS**

| ID | 统计参数的描述 | 类型 |
|---|---|---|
| STAT_INITIAL_DEPOSIT | 初始存款的值 | double |
| STAT_WITHDRAWAL | 从账户取款 | double |
| STAT_PROFIT | 测试后的净利润，STAT_GROSS_PROFIT 和 STAT_GROSS_LOSS 的总和 (STAT_GROSS_LOSS 始终小于或等于0) | double |
| STAT_GROSS_PROFIT | 总利润，所有获利（正值）交易的总和。该值大于或等于0 | double |
| STAT_GROSS_LOSS | 总亏损，所有负值交易的总和。该值小于或等于0 | double |
| STAT_MAX_PROFITTRADE | 最大收益 -所有获利交易中的最大值。该值大于或等于0 | double |
| STAT_MAX_LOSSTRADE | 最大亏损 -所有亏损交易中的最低值。该值小于或等于0 | double |
| STAT_CONPROFITMAX | 一组获利交易中的最大收益。该值大于或等于0 | double |
| STAT_CONPROFITMAX_TRADES | 已经形成 STAT_CONPROFITMAX (一组获利交易中的最大收益)的交易的数量 | int |
| STAT_MAX_CONWINS | 最长的一组获利交易的总利润 | double |
| STAT_MAX_CONPROFIT_TRADES | 最长的一组获利交易中的交易数量 STAT_MAX_CONWINS | int |

© 2000-2025, MetaQuotes Ltd.

| ID | 统计参数的描述 | 类型 |
| --- | --- | --- |
| STAT_CONLOSSMAX | 一组亏损交易中的最大亏损。该值小于或等于0 | double |
| STAT_CONLOSSMAX_TRADES | 已经形成STAT_CONLOSSMAX(一组亏损交易中的最大亏损)的交易的数量 | int |
| STAT_MAX_CONLOSSES | 最长一组亏损交易的总亏损 | double |
| STAT_MAX_CONLOSS_TRADES | 最长的一组亏损交易中的交易数量STAT_MAX_CONLOSSES | int |
| STAT_BALANCEMIN | 结余的最小值 | double |
| STAT_BALANCE_DD | 以货币计算，最大的结余跌幅。在交易处理过程中，结余可能有大幅度下跌，这里采用最大的值 | double |
| STAT_BALANCEDD_PERCENT | 以百分比计算结余下跌，记录于货币计算的最大结余跌幅的时期(STAT_BALANCE_DD)。 | double |
| STAT_BALANCE_DDREL_PERCENT | 以百分比计算最大结余跌幅。在交易处理过程中，结余可能有大幅度下跌，每次相关的下跌值都以百分比计算返回最大的值 | double |
| STAT_BALANCE_DD_RELATIVE | 以货币计算结余下跌，记录于百分比计算的最大结余跌幅时期(STAT_BALANCE_DDREL_PERCENT)。 | double |
| STAT_EQUITYMIN | 最小净值 | double |
| STAT_EQUITY_DD | 以货币计算，最大的净值跌幅。在交易处理过程中，净值上可能出现大幅度下跌，这里采用最大的值 | double |

© 2000-2025, MetaQuotes Ltd.

| ID | 统计参数的描述 | 类型 |
| --- | --- | --- |
| STAT_EQUITYDD_PERCENT | 以百分比计算下跌，记录于货币计算的最大净值跌幅的时期(STAT_EQUITY_DD)。 | double |
| STAT_EQUITY_DDREL_PERCENT | 以百分比计算最大净值跌幅。在交易处理过程中，净值可能有大幅度下跌，每次相关的下跌值都以百分比计算返回最大的值 | double |
| STAT_EQUITY_DD_RELATIVE | 以货币计算净值下跌，记录于百分比计算的最大净值跌幅时期(STAT_EQUITY_DDREL_PERCENT)。 | double |
| STAT_EXPECTED_PAYOFF | 期望收益 | double |
| STAT_PROFIT_FACTOR | 利润因子，等于STAT_GROSS_PROFIT/STAT_GROSS_LOSS比率。如果STAT_GROSS_LOSS=0，利润因子就等于DBL_MAX | double |
| STAT_RECOVERY_FACTOR | 采收率，等于STAT_PROFIT/STAT_BALANCE_DD的比率 | double |
| STAT_SHARPE_RATIO | 夏普比率 | double |
| STAT_MIN_MARGINLEVEL | 保证金水平的最小值 | double |
| STAT_CUSTOM_ONTESTER | 计算的自定义优化标准的值通过OnTester() 函数返回unction | double |
| STAT_DEALS | 成交数量 | int |
| STAT_TRADES | 交易的数量 | int |
| STAT_PROFIT_TRADES | 获利交易 | int |
| STAT_LOSS_TRADES | 亏损交易 | int |
| STAT_SHORT_TRADES | 卖出交易 | int |
| STAT_LONG_TRADES | 买入交易 | int |
| STAT_PROFIT_SHORTTRADES | 获利的卖出交易 | int |

© 2000-2025, MetaQuotes Ltd.

| ID | 统计参数的描述 | 类型 |
| --- | --- | --- |
| STAT_PROFIT_LONGTRADES | 获利的买入交易 | int |
| STAT_PROFITTRADES_AVGCON | 一组获利交易的平均长度 | int |
| STAT_LOSSTRADES_AVGCON | 一组亏损交易的平均长度 | int |
| STAT_COMPLEX_CRITERION | 复杂的优化标准 | |

© 2000-2025, MetaQuotes Ltd.

# 交易常数

用于程序买卖策略的各种常数分为如下组：

- 历史数据库属性 -接收一般信息；
- 订单属性 -获得交易订单信息；
- 仓位属性 -获得当前位置信息；
- 交易属性 -获得交易信息；
- 交易操作类型 -交易可行性描述；
- 交易事务类型 -描述可能的交易事务类型；
- DOM交易订单 -根据要求操作切割订单。

© 2000-2025, MetaQuotes Ltd.

# 历史数据库属性

当访问函数 timeseries 时，SeriesInfoInteger() 是用来获得额外 交易品种信息。必要属性标识符以函数参量传递，标识符可以是ENUM_SERIES_INFO_INTEGER值中的一个。

**ENUM_SERIES_INFO_INTEGER**

| 标识符 | 描述 | 类型 |
|---|---|---|
| SERIES_BARS_COUNT | 当前时刻交易品种周期字节价值 | long |
| SERIES_FIRSTDATE | 当前时刻交易品种周期第一数据 | datetime |
| SERIES_LASTBAR_DATE | 交易品种周期最后字节的开仓时间 | datetime |
| SERIES_SERVER_FIRSTDATE | 忽略时间表的服务器上交易品种历史中第一个日期 | datetime |
| SERIES_TERMINAL_FIRSTDATE | 忽略时间表的客户端上交易品种历史中第一个日期 | datetime |
| SERIES_SYNCHRONIZED | 当前时刻交易品种/周期的同步数据 | bool |

© 2000-2025, MetaQuotes Ltd.

# 订单属性

请求执行交易操作与命令一样正式化，每个订单都有多种属性准备，他们的信息可以使用 OrderGet...()和 HistoryOrderGet...() 函数获得。

函数 OrderGetInteger() 和 HistoryOrderGetInteger()

ENUM_ORDER_PROPERTY_INTEGER

| 标识符 | 描述 | 类型 |
|---|---|---|
| ORDER_TICKET | 订单价格。每个订单分配唯一的号码 | long |
| ORDER_TIME_SETUP | 订单设置时间 | datetime |
| ORDER_TYPE | 订单类型 | ENUM_ORDER_TYPE |
| ORDER_STATE | 订单陈述 | ENUM_ORDER_STATE |
| ORDER_TIME_EXPIRATION | 订单终结时间 | datetime |
| ORDER_TIME_DONE | 订单执行或取消时间 | datetime |
| ORDER_TIME_SETUP_MSC | 以毫秒为单位计算01.01.1970以来下订单执行的时间 | long |
| ORDER_TIME_DONE_MSC | 以毫秒为单位计算01.01.1970以来订单执行/取消的时间 | long |
| ORDER_TYPE_FILLING | 订单填满类型 | ENUM_ORDER_TYPE_FILLING |
| ORDER_TYPE_TIME | 订单使用期 | ENUM_ORDER_TYPE_TIME |
| ORDER_MAGIC | EA交易的ID安置在订单之中为确保 | long |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 每个EA交易安置都有独特的代码） | |
| ORDER_REASON | 下单的原因或源 | ENUM_ORDER_REASON |
| ORDER_POSITION_ID | 一旦执行，位置指标马上建立一个订单。每个执行订单结果都使订单公开或修改已经存在的位置，明确指出指标的位置同时执行订单 | long |
| ORDER_POSITION_BY_ID | 反向持仓标识符，用于关闭订单ORDER_TYPE_CLOSE_BY | long |

函数 OrderGetDouble() 和 HistoryOrderGetDouble()

**ENUM_ORDER_PROPERTY_DOUBLE**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| ORDER_VOLUME_INITIAL | 订单最初交易量 | double |
| ORDER_VOLUME_CURRENT | 订单当前交易量 | double |
| ORDER_PRICE_OPEN | 订单中的规定价格 | double |
| ORDER_SL | 斩仓值 | double |
| ORDER_TP | 盈利值 | double |
| ORDER_PRICE_CURRENT | 交易品种订单的当 | double |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 前价格 | |
| ORDER_PRICE_STOPLIMIT | 限制停止订单的限制价格订单 | double |

函数 OrderGetString() 和 HistoryOrderGetString()

**ENUM_ORDER_PROPERTY_STRING**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| ORDER_SYMBOL | 交易品种订单 | string |
| ORDER_COMMENT | 评价订单 | string |

当使用OrderSend()函数发送交易要求时，一些操作要求订单类型的指示，订单类型在类型列表中分类，尤其在MqlTradeRequest结构中，能接受到ENUM_ORDER_TYPE项目值。

**ENUM_ORDER_TYPE**

| 标识符 | 描述 |
| --- | --- |
| ORDER_TYPE_BUY | 市场购买订单 |
| ORDER_TYPE_SELL | 市场卖出订单 |
| ORDER_TYPE_BUY_LIMIT | 限制买入待办订单 |
| ORDER_TYPE_SELL_LIMIT | 限制卖出待办订单 |
| ORDER_TYPE_BUY_STOP | 停止买入待办订单 |
| ORDER_TYPE_SELL_STOP | 停止卖出待办订单 |
| ORDER_TYPE_BUY_STOP_LIMIT | 在到达订单价格之上，是限制买入订单安置在停止限制价格中 |
| ORDER_TYPE_SELL_STOP_LIMIT | 在到达订单价格之上，是限制卖出订单安置在停止限制价格中 |
| ORDER_TYPE_CLOSE_BY | 通过反向持仓平仓的订单 |

每一个订单都有其作用的描述，为获得信息，与ORDER_STATE修饰语一起使用OrderGetInteger()或HistoryOrderGetInteger()函数，允许值存储在ENUM_ORDER_STATE项目中。

**ENUM_ORDER_STATE**

| 标识符 | 描述 |
| --- | --- |
| ORDER_STATE_STARTED | 选中订单，并未被经纪人接收 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| ORDER_STATE_PLACED | 接收订单 |
| ORDER_STATE_CANCELED | 客户撤销订单 |
| ORDER_STATE_PARTIAL | 履行部分订单 |
| ORDER_STATE_FILLED | 履行全部订单 |
| ORDER_STATE_REJECTED | 驳回订单 |
| ORDER_STATE_EXPIRED | 过期订单 |
| ORDER_STATE_REQUEST_ADD | 注册订单（到交易系统） |
| ORDER_STATE_REQUEST_MODIFY | 更改订单（改变其参数） |
| ORDER_STATE_REQUEST_CANCEL | 删除订单（从交易系统中删除） |

当**在当前时间**（有效时间）发送执行交易请求时，应该指定价格和所需的买卖交易量。此外，请记住，金融市场并不能保证某一交易品种可以期望的价格获得所需的全部交易量。因此，实时交易操作通过价格和交易量执行模式进行管理。模式或执行指令定义了价格改变或请求交易量不能在**此刻**完全满足的情况下的规则。

**价格执行模式**可以从SYMBOL_TRADE_EXEMODE交易品种属性中获得，其包含来自ENUM_SYMBOL_TRADE_EXECUTION枚举的标识组合。

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| 执行模式<br><br>（要价执行） | 以之前从交易商接收的价格执行市价单。<br><br>在订单发送之前，交易商会请求特定市价单的价格。在收到价格 | SYMBOL_TRADE_EXECUTION_REQUEST |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| | 后，可以确认或拒绝以给定价格执行订单。 | |
| 现价执行<br><br>(现价执行) | 立即以指定价格执行市价单。<br><br>当发送要执行的交易请求时，平台会自动将当前价格添加到订单中。<br>• 如果交易商接受该价格，则订单执行。 | SYMBOL_TRADE_EXECUTION_INSTANT |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| | • 如果交易商不接受请求价格，则发送"Requote"（重报价）—交易商返回可以执行此订单的价格。 | |

© 2000-2025, MetaQuotes Ltd.

| 执行模式 | 描述 | ENUM_SYMBOL_TRADE_EXECUTION中的值 |
| --- | --- | --- |
| 市价执行<br><br>(市价执行) | 交易商不需要与交易者进行额外商量就可以决定订单执行价格。<br><br>以这种模式发送订单意味着事先同意以这个价格执行。 | SYMBOL_TRADE_EXECUTION_MARKET |
| 交易所执行<br><br>(交易所执行) | 交易操作以当前市场报价执行。 | SYMBOL_TRADE_EXECUTION_EXCHANGE |

交易量成交指令在ORDER_TYPE_FILLING订单属性中指定，可能只包含ENUM_ORDER_TYPE_FILLING枚举值

| 成交指令 | 描述 | ENUM_ORDER_TYPE_FILLING值 |
| --- | --- | --- |
| 全数执行或立即取消 | 订单只能按照指定的交易量执行。 | ORDER_FILLING_FOK |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | 描述 | ENUM_ORDER_TYPE_FILLING值 |
| --- | --- | --- |
| | 如果交易品种的所需交易量目前在市场上不能满足，则该订单将不会被执行。<br><br>所需交易量可以由几个可用报价组成。<br><br>由交易服务器决定使用FOK订单的可能性。 | |
| 立即或取消 | 交易者同意按照订单中指定的场内最大交易量执行成交。 | ORDER_FILLING_IOC |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | 描述 | ENUM_ORDER_TYPE_FILLING值 |
| --- | --- | --- |
| | 如果不能完全成交，则将执行可用交易量的订单，并取消其余交易量。<br><br>由交易服务器决定使用IOC订单的可能性。 | |
| 被动（预订或取消） | BoC订单假设订单只能在市场深度下单，不能立即执行。如果订单可以在下单后立即执行，则取消订单。<br><br>事实上， | ORDER_FILLING_BOC |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | 描述 | ENUM_ORDER_TYPE_FILLING值 |
| --- | --- | --- |
| | BOC指令保证了所下订单的价格会不及当前市场的价格。BoC订单用于实施被动交易，因此订单在下单时不会立即执行，也不会影响当前的流动性。<br><br>仅支持限价单和止损限价单(ORDER_TYPE_BUY_LIMIT、ORDER_TYPE_SELL_LIMIT、ORDER_TYPE | |

© 2000-2025, MetaQuotes Ltd.

| 成交指令 | 描述 | ENUM_ORDER_TYPE_FILLING值 |
| --- | --- | --- |
| | _BUY_STOP_LIMIT、ORDER_TYPE_SELL_STOP_LIMIT)。 | |
| 返回 | 如果部分成交，则订单剩余交易量不会取消，会进一步处理。<br><br>在市价执行模式下不允许返回订单（市价执行 — SYMBOL_TRADE_EXECUTION_MARKET）。 | ORDER_FILLING_RETURN |

当使用OrderSend()函数发送交易请求时，所需交易量执行指令可以在*type_filling*字段中设置，即在特殊的MqlTradeRequest结构中设置。可以使用ENUM_ORDER_TYPE_FILLING枚举值。若要获取特定活动/完成订单中的属性值，请使用OrderGetInteger()或HistoryOrderGetInteger()函数（带有ORDER_TYPE_FILLING修饰符）。

在发送当前执行时间的订单之前，为了正确设置ORDER_TYPE_FILLING值（交易量执行类型），您可以对每个交易品种使用SymbolInfoInteger()函数来获取 SYMBOL_FILLING_MODE属性值，将交易

© 2000-2025, MetaQuotes Ltd.

品种允许的交易量执行类型显示为标识组合。除了“市价执行”模式(SYMBOL_TRADE_EXECUTION_MARKET)外，将一直启用ORDER_FILLING_RETURN成交类型。

根据执行模式使用的成交类型可以如下表所示：

| 执行类型/成交指令类型 | 全数执行或立即取消(FOK ORDER_FILLING_FOK) | 立即或取消(IOC ORDER_FILLING_IOC) | 返回(返回ORDER_FILLING_RETURN) |
|---|---|---|---|
| 现价执行<br><br>(SYMBOL_TRADE_EXECUTION_INSTANT) | +（无论交易品种如何设置） | +（无论交易品种如何设置） | +（始终） |
| 要价执行<br><br>SYMBOL_TRADE_EXECUTION_REQUEST | +（无论交易品种如何设置） | +（无论交易品种如何设置） | +（始终） |
| 市价执行<br><br>SYMBOL_TRADE_EXECUTION_MARKET | +（在交易品种设置中设置） | +（在交易品种设置中设置） | -（无论交易品种如何设置，请禁用） |
| 交易所执行<br><br>SYMBOL_TRADE_EXECUTION_EXCHANGE | +（在交易品种设置中设置） | +（在交易品种设置中设置） | +（始终） |

挂单情况下，无论执行类型(SYMBOL_TRADE_EXEMODE)如何，都应该使用ORDER_FILLING_RETURN成交类型，因为这类订单不会在发送时执行。使用挂单时，交易者事先同意，当满足此订单的交易条件时，交易商将使用交易所支持的成交类型。

当使用OrderSend()函数发送交易请求时，订单有效期设定在MqlTradeRequest特殊结构中的*type_time*领域，ENUM_ORDER_TYPE_TIME项目值是允许的，使用ORDER_TYPE_TIME修饰语和OrderGetInteger()或HistoryOrderGetInteger()函数可以获得该性质的值。

**ENUM_ORDER_TYPE_TIME**

| 标识符 | 描述 |
|---|---|
| ORDER_TIME_GTC | 取消订单 |
| ORDER_TIME_DAY | 前交易日订单 |
| ORDER_TIME_SPECIFIED | 过期订单 |
| ORDER_TIME_SPECIFIED_DAY | 订单将生效直至指定日期的23:59:59。如果该时间超出了交易期，那么该订单会在最近的交易时间内到期。 |

下单原因包含在ORDER_REASON属性之中。可以通过MQL5程序，移动程序端以及StopOut来下单。可能的ORDER_REASON值会在ENUM_ORDER_REASON枚举中描述。

**ENUM_ORDER_REASON**

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| ORDER_REASON_CLIENT | 从桌面程序端下单 |
| ORDER_REASON_MOBILE | 从移动程序端下单 |
| ORDER_REASON_WEB | 从网页平台下单 |
| ORDER_REASON_EXPERT | 从MQL5程序下单，例如EA交易或脚本 |
| ORDER_REASON_SL | 因激活止损而下单 |
| ORDER_REASON_TP | 因激活止赢而下单 |
| ORDER_REASON_SO | 因Stop Out事件而下单 |

© 2000-2025, MetaQuotes Ltd.

# 仓位属性

在开盘中执行交易操作结果，改变成交量和/或位置，或者消失。交易操作以 订单为基础管理，在交易请求 形式中通过 OrderSend() 函数发送，对于每一个金融 证券 (symbol) 交易品种)来说，一个开仓位置是可行的，通过 PositionGet...() 函数一个位置有一个属性准备设立。

函数 PositionGetInteger()

ENUM_POSITION_PROPERTY_INTEGER

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| POSITION_TICKET | 持仓价格。每个新持仓分配唯一的号码通常用于持仓的订单价格是匹配的，除非由于服务器的服务操作而导致价格发生变化，例如，收取重新持仓的库存费时。若要寻找用于持仓的订单，应用 POSITION_IDENTIFIER 属性。<br><br>POSITION_TICKET 值相对应 MqlTradeRequest::position。 | long |
| POSITION_TIME | 开盘时间位置 | datetime |
| POSITION_TIME_MSC | 以毫秒为单位计算01.01.1970以来持仓的时间 | long |
| POSITION_TIME_UPDATE | 以秒为单位计算01.01.1970以来持仓更改的时间 | datetime |
| POSITION_TIME_UPDATE_MSC | 以毫秒为单位计算01.01.1970以来持仓更改的时间 | long |
| POSITION_TYPE | 位置类型 | ENUM_POSITION_TYPE |
| POSITION_MAGIC | 位置幻数 (参看 ORDER_MAGIC ) | long |
| POSITION_IDENTIFIER | 位置标识符是独一的数字，对于 | long |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | 每个新的开仓位来说是制定的且在整个使用期内是不可更改的，翻转位置不能改变它的标识符。每个订单(ORDER_POSITION_ID)和每笔交易(DEAL_POSITION_ID) 都会指定用来打开、更改或关闭它的持仓识别器。使用这个属性来搜索持仓相关的订单和交易。当单边模式下逆转持仓 (使用单进/单出交易）时，POSITION_IDENTIFIER不会改变。但是，POSITION_TICKET会被替换为导致逆转的订单单号。而在锁仓模式下不提供逆转持仓。 | |
| POSITION_REASON | 持仓的原因 | ENUM_POSITION_REASON |

函数 PositionGetDouble()

ENUM_POSITION_PROPERTY_DOUBLE

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| POSITION_VOLUME | 方位成交量 | double |
| POSITION_PRICE_OPEN | 方位开盘价格 | double |
| POSITION_SL | 开仓止损水平 | double |
| POSITION_TP | 开仓获利水平 | double |
| POSITION_PRICE_CURRENT | 开仓当前价位 | double |
| POSITION_SWAP | 积累交换 | double |
| POSITION_PROFIT | 当前利润 | double |

© 2000-2025, MetaQuotes Ltd.

函数 PositionGetString()

ENUM_POSITION_PROPERTY_STRING

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| POSITION_SYMBOL | 交易品种位置 | string |
| POSITION_COMMENT | 方位注释 | string |
| POSITION_EXTERNAL_ID | 外部交易系统（交易所）中的持仓ID | string |

开仓指向（买或卖）通过ENUM_POSITION_TYPE值来定义。为了获得开仓类型，使用PositionGetInteger()函数POSITION_TYPE修饰语。

ENUM_POSITION_TYPE

| 标识符 | 描述 |
| --- | --- |
| POSITION_TYPE_BUY | 买入 |
| POSITION_TYPE_SELL | 卖出 |

持仓的原因包含在POSITION_REASON属性之中。因激活桌面程序端，移动程序端，EA交易等的下单来持仓。可能的POSITION_REASON值会在ENUM_POSITION_REASON枚举中描述。

ENUM_POSITION_REASON

| 标识符 | 描述 |
| --- | --- |
| POSITION_REASON_CLIENT | 因激活桌面程序端的下单来持仓 |
| POSITION_REASON_MOBILE | 因激活移动程序端的下单来持仓 |
| POSITION_REASON_WEB | 因激活网页平台的下单来持仓 |
| POSITION_REASON_EXPERT | 因激活MQL5程序的下单来持仓，例如EA交易或脚本 |

© 2000-2025, MetaQuotes Ltd.

# 交易属性

一个交易反映了以订单为基础的交易操作 执行，包括交易要求。每个交易都通过属性描述来获得有关信息，为了获取属性值，使用 HistoryDealGet...() 函数类型，从类似项目中获得返回值。

函数 HistoryDealGetInteger()

ENUM_DEAL_PROPERTY_INTEGER

| 标识符 | 描述 | 类型 |
|---|---|---|
| DEAL_TICKET | 交易标签。分配到每笔交易的唯一号码 | long |
| DEAL_ORDER | 交易 订单号 | long |
| DEAL_TIME | 交易时间 | datetime |
| DEAL_TIME_MSC | 以毫秒为单位计算01.01.1970以来交易执行的时间 | long |
| DEAL_TYPE | 交易类型 | ENUM_DEAL_TYPE |
| DEAL_ENTRY | 交易条目—允许进入、允许输出、颠倒 | ENUM_DEAL_ENTRY |
| DEAL_MAGIC | 订单幻数（参见 ORDER_MAGIC） | long |
| DEAL_REASON | 交易执行的原因或源 | ENUM_DEAL_REASON |
| DEAL_POSITION_ID | 仓位标识符在该交易持仓、修正或者平仓时，每个位置都有一个独特的标识符，为交易对象它被委派到所有执行订单，在全部方位使用期 | long |

函数 HistoryDealGetDouble()

ENUM_DEAL_PROPERTY_DOUBLE

| 标识符 | 描述 | 类型 |
|---|---|---|
| DEAL_VOLUME | 成交量 | double |
| DEAL_PRICE | 成交价格 | double |
| DEAL_COMMISSION | 订单佣金关 | double |
| DEAL_SWAP | 关闭的积累交换 | double |
| DEAL_PROFIT | 交易利润 | double |
| DEAL_FEE | 执行交易之后立即收取的交易费用 | double |
| DEAL_SL | 止损水平<br>• 进场交易和反转交易使用原始订单中的止损值，基于该值进行开仓或反向开仓。 | 双精度 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| | • 出场交易使用平仓时持仓的止损值 | |
| DEAL_TP | 止盈水平<br>• 进场交易和反转交易使用原始订单中的止盈值，基于该值进行开仓或反向开仓<br>• 出场交易使用平仓时持仓的止盈值 | 双精度 |

函数 HistoryDealGetString()

**ENUM_DEAL_PROPERTY_STRING**

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| DEAL_SYMBOL | 订单交易品种 | string |
| DEAL_COMMENT | 订单佣金 | string |

每笔订单都具有类型特征，允许在ENUM_DEAL_TYPE中列举价值。为了获得有关订单类型的信息，使用 HistoryDealGetInteger() 函数DEAL_TYPE 修饰语。

**ENUM_DEAL_TYPE**

| 标识符 | 描述 |
| --- | --- |
| DEAL_TYPE_BUY | 买入 |
| DEAL_TYPE_SELL | 卖出 |
| DEAL_TYPE_BALANCE | 结算 |
| DEAL_TYPE_CREDIT | 信贷 |
| DEAL_TYPE_CHARGE | 额外金额 |
| DEAL_TYPE_CORRECTION | 修正 |
| DEAL_TYPE_BONUS | 奖金 |
| DEAL_TYPE_COMMISSION | 额外手续费 |
| DEAL_TYPE_COMMISSION_DAILY | 日手续费 |
| DEAL_TYPE_COMMISSION_MONTHLY | 月手续费 |
| DEAL_TYPE_COMMISSION_AGENT_DAILY | 日代理手续费 |
| DEAL_TYPE_COMMISSION_AGENT_MONTHLY | 月代理手续费 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| DEAL_TYPE_INTEREST | 利率 |
| DEAL_TYPE_BUY_CANCELED | 取消的买入交易。可能会有之前执行的买入交易被取消的情况。这样的话，之前执行的交易类型 (DEAL_TYPE_BUY) 会被改为DEAL_TYPE_BUY_CANCELED，并且其利润/亏损归零。执行获得的利润/亏损会使用单独的结余操作被收费/取款。 |
| DEAL_TYPE_SELL_CANCELED | 取消的卖出交易。可能会有之前执行的卖出交易被取消的情况。这样的话，之前执行的交易类型 (DEAL_TYPE_SELL)会被改为DEAL_TYPE_SELL_CANCELED，并且其利润/亏损归零。执行获得的利润/亏损会使用单独的结余操作被收费/取款。 |
| DEAL_DIVIDEND | 股息操作 |
| DEAL_DIVIDEND_FRANKED | 税务减免（免税）股息操作 |
| DEAL_TAX | 税费 |

订单不仅与建立在ENUM_DEAL_TYPE中的类型不同，也与改变方位的方式不同，这是一个简单的位置开仓，或者先前开仓的聚集（进入市场），如果相反方向订单覆盖了先前的开仓位置成交量，通过相似成交量（退出市场）的反向订单关闭位置。

所有情况在ENUM_DEAL_ENTRY值中都有描述。为了获得订单信息，使用HistoryDealGetInteger()函数DEAL_ENTRY修饰语。

**ENUM_DEAL_ENTRY**

| 标识符 | 描述 |
| --- | --- |
| DEAL_ENTRY_IN | 进入 |
| DEAL_ENTRY_OUT | 出局 |
| DEAL_ENTRY_INOUT | 反面 |
| DEAL_ENTRY_OUT_BY | 通过反向持仓来平仓 |

交易执行的原因包含在DEAL_REASON属性之中。触发从移动程序端或MQL5程序下的订单，以及通过StopOut事件，变动预付款计算等等可以执行交易。可能的DEAL_REASON值会在ENUM_DEAL_REASON枚举中描述。对于来自结余，信贷，手续费和其他操作的非交易成交，DEAL_REASON_CLIENT会表示为原因。

**ENUM_DEAL_REASON**

| 标识符 | 描述 |
| --- | --- |
| DEAL_REASON_CLIENT | 因激活桌面程序端下单的订单来执行交易 |
| DEAL_REASON_MOBILE | 因激活移动程序端下单的订单来执行交易 |
| DEAL_REASON_WEB | 因激活网页平台下单的订单来执行交易 |
| DEAL_REASON_EXPERT | 因激活从MQL5程序下单的订单来执行交易，例如EA交易或脚本 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| DEAL_REASON_SL | 因激活止损来执行交易 |
| DEAL_REASON_TP | 因激活止赢来执行交易 |
| DEAL_REASON_SO | 因Stop Out事件来执行交易 |
| DEAL_REASON_ROLLOVER | 因rollover来执行交易 |
| DEAL_REASON_VMARGIN | 收取变动预付款后执行交易 |
| DEAL_REASON_SPLIT | 宣布分割期间拥有持仓的品种进行分割（价格降低）后执行交易 |
| DEAL_REASON_CORPORATE_ACTION | 该交易的执行源于公司行为：合并或重命名证券、将客户转移到另一个账户等 |

© 2000-2025, MetaQuotes Ltd.

# 交易操作类型

使用OrderSend()函数通过向开仓位置发送命令进行交易，和放置、修改或删除待办订单命令一样。每个交易命令引用要求操作类型。交易操作在ENUM_TRADE_REQUEST_ACTIONS项目中描述。

**ENUM_TRADE_REQUEST_ACTIONS**

| 标识符 | 描述 |
| --- | --- |
| TRADE_ACTION_DEAL | 为规定参数的立即执行放置交易命令（市场命令） |
| TRADE_ACTION_PENDING | 在制定环境下执行放置交易命令（待办订单） |
| TRADE_ACTION_SLTP | 修改折仓并取走开仓利润值 |
| TRADE_ACTION_MODIFY | 修改先前放置的命令参量 |
| TRADE_ACTION_REMOVE | 删除先前放置的待办订单命令 |
| TRADE_ACTION_CLOSE_BY | 通过反向持仓来平仓 |

打开买入持仓TRADE_ACTION_DEAL交易操作的示例：

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 打开买入持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 请求的参数
   request.action   =TRADE_ACTION_DEAL;                     // 交易操作类型
   request.symbol   =Symbol();                              // 交易品种
   request.volume   =0.1;                                   // 0.1手交易量
   request.type     =ORDER_TYPE_BUY;                        // 订单类型
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // 持仓价格
   request.deviation=5;                                     // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                          // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // 如果不能发送请求，输出错误代
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

打开卖出持仓TRADE_ACTION_DEAL交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 打开卖出持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 请求参数
   request.action   =TRADE_ACTION_DEAL;                       // 交易操作类型
   request.symbol   =Symbol();                                // 交易品种
   request.volume   =0.2;                                     // 0.2 手交易量
   request.type     =ORDER_TYPE_SELL;                         // 订单类型
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_BID); // 持仓价格
   request.deviation=5;                                       // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                            // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // 如果不能发送请求，输出错误代
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

平仓TRADE_ACTION_DEAL交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 关闭全部持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // 持仓数
//--- 重做所有持仓
   for(int i=total-1; i>=0; i--)
     {
      //--- 订单的参数
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- 输出持仓信息
      PrintFormat("#%I64u %s  %s  %.2f  %s [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  magic);
      //--- 如果幻数匹配
      if(magic==EXPERT_MAGIC)
        {
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置操作参数
         request.action   =TRADE_ACTION_DEAL;        // 交易操作类型
         request.position =position_ticket;          // 持仓价格
         request.symbol   =position_symbol;          // 交易品种
         request.volume   =volume;                   // 持仓交易量
         request.deviation=5;                        // 允许价格偏差
         request.magic    =EXPERT_MAGIC;             // 持仓幻数
         //--- 根据持仓类型设置价格和订单类型
         if(type==POSITION_TYPE_BUY)
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
            request.type =ORDER_TYPE_SELL;
           }
         else
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
            request.type =ORDER_TYPE_BUY;
           }
         //--- 输出关闭信息
         PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToString
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //---
        }
```

© 2000-2025, MetaQuotes Ltd.

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

下挂单 TRADE_ACTION_PENDING 交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456                                  // EA交易的幻数
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;    // 订单类型
//+------------------------------------------------------------------+
//| 下挂单                                                           |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 下挂单的参数
   request.action   =TRADE_ACTION_PENDING;                                  // 交易操作类型
   request.symbol   =Symbol();                                              // 交易品种
   request.volume   =0.1;                                                   //0.1手交易量
   request.deviation=2;                                                     // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                                          // 订单幻数
   int offset = 50;                                                         // 以点数从当前价
   double price;                                                            // 订单触动价
   double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT);                     // value of poi
   int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);                     // 小数位数（精确
   //--- 检查操作类型
   if(orderType==ORDER_TYPE_BUY_LIMIT)
     {
      request.type     =ORDER_TYPE_BUY_LIMIT;                               // 订单类型
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;             // 持仓价格
      request.price    =NormalizeDouble(price,digits);                      //正常开盘价
     }
   else if(orderType==ORDER_TYPE_SELL_LIMIT)
     {
      request.type     =ORDER_TYPE_SELL_LIMIT;                               // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;              // 持仓价
      request.price    =NormalizeDouble(price,digits);                       // 正常开盘价
     }
   else if(orderType==ORDER_TYPE_BUY_STOP)
     {
      request.type =ORDER_TYPE_BUY_STOP;                                     // 订单类型
      price        =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;      // 开盘价
      request.price=NormalizeDouble(price,digits);                           // 正常开盘价
     }
   else if(orderType==ORDER_TYPE_SELL_STOP)
     {
      request.type     =ORDER_TYPE_SELL_STOP;                                // 订单类型
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;              // 开盘价
      request.price    =NormalizeDouble(price,digits);                       // 正常开盘价
     }
   else Alert("This example is only for placing pending orders");    // 如果没有挂单被选择
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());                      // 如果不能发送
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

更改持仓止损和止赢值的 TRADE_ACTION_SLTP 交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 更改持仓的止损和止赢                                             |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // 持仓数
//--- 重做所有持仓
   for(int i=0; i<total; i++)
     {
      //--- 订单的参数
      ulong  position_ticket=PositionGetTicket(i);// 持仓价格
      string position_symbol=PositionGetString(POSITION_SYMBOL); // 交易品种
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // 小数位数
      ulong  magic=PositionGetInteger(POSITION_MAGIC); // 持仓的幻数
      double volume=PositionGetDouble(POSITION_VOLUME);    // 持仓交易量
      double sl=PositionGetDouble(POSITION_SL);  // 持仓止损
      double tp=PositionGetDouble(POSITION_TP);  // 持仓止赢
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- 输出持仓信息
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- 如果幻数匹配，不定义止损和止赢
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 计算当前价格水平
         double price=PositionGetDouble(POSITION_PRICE_OPEN);
         double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
         double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
         int    stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_L
         double price_level;
         //--- 如果最小值被接受，那么当前平仓价的点数偏距不设置
         if(stop_level<=0)
            stop_level=150; // 设置当前平仓价的150点偏距
         else
            stop_level+=50; // 为了可靠性而设置偏距到(SYMBOL_TRADE_STOPS_LEVEL + 50) 点

         //--- 计算并凑整止损和止赢值
         price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
         if(type==POSITION_TYPE_BUY)
           {
            sl=NormalizeDouble(bid-price_level,digits);
            tp=NormalizeDouble(ask+price_level,digits);
           }
         else
           {
            sl=NormalizeDouble(ask+price_level,digits);
            tp=NormalizeDouble(bid-price_level,digits);
           }
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置操作参数
         request.action  =TRADE_ACTION_SLTP; // 交易操作类型
         request.position=position_ticket;   // 持仓价格
         request.symbol=position_symbol;     // 交易品种
         request.sl      =sl;                // 持仓止损
         request.tp      =tp;                // 持仓止赢
         request.magic=EXPERT_MAGIC;         // 持仓的幻数
         //--- 输出更改信息
         PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToStrin
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

更改挂单价格水平的TRADE_ACTION_MODIFY交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 更改挂单                                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // 已下挂单的总数
//--- 重做所有已下的挂单
   for(int i=0; i<total; i++)
     {
      //--- 订单参数
      ulong  order_ticket=OrderGetTicket(i);                                  // 订单价格
      string order_symbol=Symbol();                                           // 交易品种
      int    digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS);       // 小数位数
      ulong  magic=OrderGetInteger(ORDER_MAGIC);                              // 订单的幻数
      double volume=OrderGetDouble(ORDER_VOLUME_CURRENT);                     // 订单的当前交
      double sl=OrderGetDouble(ORDER_SL);                                     // 订单的当前止
      double tp=OrderGetDouble(ORDER_TP);                                     // 订单的当前止
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE);      // 订单类型
      int offset = 50;                                                        // 以点数从当前
      double price;                                                           // 订单触动价
      double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT);               // 点值
      //--- 输出订单信息
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  order_ticket,
                  order_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- 如果幻数匹配，不定义止损和止赢
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         request.action=TRADE_ACTION_MODIFY;                                  // 交易操作类型
         request.order = OrderGetTicket(i);                                   // 订单价格
         request.symbol    =Symbol();                                         // symbol
         request.deviation=5;                                                 // 允许价格偏差
        //--- 根据持仓类型设置价格水平，订单的止损和止赢
         if(type==ORDER_TYPE_BUY_LIMIT)
           {
            price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开盘
           }
         else if(type==ORDER_TYPE_SELL_LIMIT)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         else if(type==ORDER_TYPE_BUY_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
```

© 2000-2025, MetaQuotes Ltd.

```
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         else if(type==ORDER_TYPE_SELL_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
        }
     }
  }
//+------------------------------------------------------------------+
```

删除挂单的TRADE_ACTION_REMOVE交易操作的示例：

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 删除挂单                                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // 下挂单的总数
//--- 重做所有已下的挂单
   for(int i=total-1; i>=0; i--)
     {
      ulong  order_ticket=OrderGetTicket(i);                   // order ticket
      ulong  magic=OrderGetInteger(ORDER_MAGIC);               // 订单的幻数
      //--- 如果幻数匹配
      if(magic==EXPERT_MAGIC)
        {
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置持仓参数
         request.action=TRADE_ACTION_REMOVE;                   // 交易操作的类型
         request.order = order_ticket;                         // 订单价格
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

关闭反向持仓的TRADE_ACTION_CLOSE_BY交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 关闭所有反向持仓                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // 持仓数
//--- 重做所有持仓
   for(int i=total-1; i>=0; i--)
     {
      //--- 订单的参数
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      double sl=PositionGetDouble(POSITION_SL);
      double tp=PositionGetDouble(POSITION_TP);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- 输出持仓信息
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- 如果幻数匹配
      if(magic==EXPERT_MAGIC)
        {
         for(int j=0; j<i; j++)
           {
            string symbol=PositionGetSymbol(j); // 反向持仓交易品种
            //--- 如果反向持仓交易品种和初始交易品种匹配
            if(symbol==position_symbol && PositionGetInteger(POSITION_MAGIC)==EXPERT_M
              {
               //--- 设置反向持仓类型
               ENUM_POSITION_TYPE type_by=(ENUM_POSITION_TYPE)PositionGetInteger(POSIT
               //--- 离开，如果初始持仓和反向持仓类型匹配
               if(type==type_by)
                  continue;
               //--- 归零请求和结果值
               ZeroMemory(request);
               ZeroMemory(result);
               //--- 设置操作参数
               request.action=TRADE_ACTION_CLOSE_BY;                          // 交易操
               request.position=position_ticket;                              // 持仓价
               request.position_by=PositionGetInteger(POSITION_TICKET);       // 反向持
               //request.symbol     =position_symbol;
               request.magic=EXPERT_MAGIC;                                    // 持仓的
               //--- 输出反向持仓平仓的信息
               PrintFormat("Close #%I64d %s %s by #%I64d",position_ticket,position_sym
               //--- 发送请求
               if(!OrderSend(request,result))
                  PrintFormat("OrderSend error %d",GetLastError()); // 如果不能发送请求,
```

© 2000-2025, MetaQuotes Ltd.

```
                //--- 操作信息
                PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result
               }
            }
         }
      }
   }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# 交易事务类型

当在交易账户上执行一些明确活动时，它的状态会发生变化。这种活动包括：

- 使用 OrderSend 和 OrderSendAsync 函数在客户端从任何MQL5应用发送交易请求及其进一步执行；
- 通过程序端图形界面发送交易请求及其进一步执行；
- 在服务器上激活挂单和止损订单；
- 在交易服务器上执行操作。

这些操作的结果会执行以下交易事务：

- 处理交易请求；
- 改变持仓订单；
- 改变订单历史记录；
- 改变交易历史记录；
- 改变持仓。

例如，当发送市场买入订单时，它会被处理，为账户创建一个适当的买入订单，然后执行订单并从持仓订单列表中移除，然后它被添加到订单历史记录，相应的交易也被添加到历史记录并创建新持仓。所有这些操作都是交易操作。

若要让程序员跟踪交易账户相关执行的操作，提供了OnTradeTransaction 函数。该处理程序允许交易事务应用到MQL5应用程序中的账户。交易事务描述使用MqlTradeTransaction 结构以OnTradeTransaction第一个参数提交。

交易事务类型以MqlTradeTransaction结构的类型参数提交。可能的交易事务类型通过以下枚举予以说明：

**ENUM_TRADE_TRANSACTION_TYPE**

| 标识符 | 描述 |
| --- | --- |
| TRADE_TRANSACTION_ORDER_ADD | 新增新的持仓订单。 |
| TRADE_TRANSACTION_ORDER_UPDATE | 更新持仓订单。更新不仅包括明显的客户端或交易服务器的变化还包括设置时的订单状态的变化（例如，转换从 ORDER_STATE_STARTED 到 ORDER_STATE_PLACED 或从 ORDER_STATE_PLACED 到 ORDER_STATE_PARTIAL等等）。 |
| TRADE_TRANSACTION_ORDER_DELETE | 从持仓订单列表移除订单。设置相应的请求或执行（填充）和移动到历史记录后，订单可以从持仓订单中删除。 |
| TRADE_TRANSACTION_DEAL_ADD | 添加交易到历史记录。订单执行执行操作或者执行账户结余操作。 |
| TRADE_TRANSACTION_DEAL_UPDATE | 更新历史记录交易。可能会有之前执行的交易变更到服务器上的情况。例如，交易在之前交易商转移的外部交易系统（交换）中已经发生了变化。 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| TRADE_TRANSACTION_DEAL_DELETE | 删除历史记录的交易。可能会有之前执行的交易从服务器删除的情况。例如，交易在之前交易商转移的外部交易系统（交换）中已经被删除。 |
| TRADE_TRANSACTION_HISTORY_ADD | 添加订单到历史记录就是执行或取消的结果。 |
| TRADE_TRANSACTION_HISTORY_UPDATE | 改变位于订单历史中的订单。这个类型用于加强交易服务器的功能。 |
| TRADE_TRANSACTION_HISTORY_DELETE | 删除订单历史记录的订单。这个类型用于加强交易服务器的功能。 |
| TRADE_TRANSACTION_POSITION | 改变与交易执行无关的持仓。这种事务类型表示持仓可以在交易服务器上改变。持仓交易量，开盘价，止损和获利水平可以改变。更改的数据以 MqlTradeTransaction 结构通过 OnTradeTransaction 处理器来提交。持仓变化（添加，更改或关闭），作为交易执行的结果，不会引起 TRADE_TRANSACTION_POSITION 事务的出现。 |
| TRADE_TRANSACTION_REQUEST | 服务器处理交易请求和处理结果的通知已经收到。只有类型字段（交易事务类型）必须以 MqlTradeTransaction 结构分析该事务。OnTradeTransaction (请求和结果)的第二和第三参数必须分析额外的数据。 |

根据交易事务类型，多种参数填写MqlTradeTransaction结构描述它。详细的提交数据描述显示在"交易事务结构"。

另见

交易事务结构, OnTradeTransaction

© 2000-2025, MetaQuotes Ltd.

# 市场深度的交易订单

对股票来说，深度市场窗口是可行的，你能看到当前买入和卖出命令。交易操作的期望目标，要求数量和要求价格对每个命令都是制定的。

通过MQL5为获得现行状态的DOM信息，可以使用 MarketBookGet() 函数，它在DOM 目录"screen shot"下的MqlBookInfo 结构数组中。

**ENUM_BOOK_TYPE**

| 标识符 | 描述 |
| --- | --- |
| BOOK_TYPE_SELL | 卖出命令 (要价) |
| BOOK_TYPE_BUY | 买入命令 (出价) |
| BOOK_TYPE_SELL_MARKET | 市场的卖出订单 |
| BOOK_TYPE_BUY_MARKET | 市场的买入订单 |

另见

结构和类， DOM结构， 交易操作类型， 市场信息

© 2000-2025, MetaQuotes Ltd.

# 信号属性

使用交易信号和信号复制设置时会使用以下枚举。

double交易信号类型属性的枚举：

**ENUM_SIGNAL_BASE_DOUBLE**

| ID | 描述 |
|---|---|
| SIGNAL_BASE_BALANCE | 账户结余 |
| SIGNAL_BASE_EQUITY | 账户净值 |
| SIGNAL_BASE_GAIN | 账户所得 |
| SIGNAL_BASE_MAX_DRAWDOWN | 账户最大跌幅 |
| SIGNAL_BASE_PRICE | 信号订阅价格 |
| SIGNAL_BASE_ROI | 投资利润率 (%) |

integer 交易信号类型属性的枚举：

**ENUM_SIGNAL_BASE_INTEGER**

| ID | 描述 |
|---|---|
| SIGNAL_BASE_DATE_PUBLISHED | 发布日期 (提供订阅的日期) |
| SIGNAL_BASE_DATE_STARTED | 监控起始日期 |
| SIGNAL_BASE_DATE_UPDATED | 信号交易统计的最近更新日期 |
| SIGNAL_BASE_ID | 信号 ID |
| SIGNAL_BASE_LEVERAGE | 账户杠杆 |
| SIGNAL_BASE_PIPS | 利润点 |
| SIGNAL_BASE_RATING | 持仓评级 |
| SIGNAL_BASE_SUBSCRIBERS | 订阅人数量 |
| SIGNAL_BASE_TRADES | 交易数量 |
| SIGNAL_BASE_TRADE_MODE | 账户类型 (0-真实，1-模拟，2-竞赛) |

string 交易信号类型属性的枚举：

**ENUM_SIGNAL_BASE_STRING**

| ID | 描述 |
|---|---|
| SIGNAL_BASE_AUTHOR_LOGIN | 作者登录 |
| SIGNAL_BASE_BROKER | 交易商名称 (公司名) |
| SIGNAL_BASE_BROKER_SERVER | 交易商服务器 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| SIGNAL_BASE_NAME | 信号名称 |
| SIGNAL_BASE_CURRENCY | 信号基础货币 |

double 信号复制设置类型属性的枚举：

**ENUM_SIGNAL_INFO_DOUBLE**

| ID | 描述 |
| --- | --- |
| SIGNAL_INFO_EQUITY_LIMIT | 净值限制 |
| SIGNAL_INFO_SLIPPAGE | 滑移(用于下市价单时同步持仓和复制交易) |
| SIGNAL_INFO_VOLUME_PERCENT | 最大的使用入金百分比(%), r/o |

integer 信号复制设置类型属性的枚举：

**ENUM_SIGNAL_INFO_INTEGER**

| ID | 描述 |
| --- | --- |
| SIGNAL_INFO_CONFIRMATIONS_DISABLED | 确认禁用标识 |
| SIGNAL_INFO_COPY_SLTP | 复制止损和获利标识 |
| SIGNAL_INFO_DEPOSIT_PERCENT | 入金比例 (%) |
| SIGNAL_INFO_ID | 信号 id, r/o |
| SIGNAL_INFO_SUBSCRIPTION_ENABLED | 订阅启用标识 |
| SIGNAL_INFO_TERMS_AGREE | "同意信号服务使用条款"标识, r/o |

string 信号复制设置类型属性的枚举：

**ENUM_SIGNAL_INFO_STRING**

| ID | 描述 |
| --- | --- |
| SIGNAL_INFO_NAME | 信号名称，r/o |

另见

交易信号

© 2000-2025, MetaQuotes Ltd.

# 命名常量

所有使用在MQL5中的常量都能分成如下几组：

- 预定义大量代替值 -在编辑过程中代替价值
- 数学常量 -一些数学表达式的值
- 数值类型常量 -一些简单类型限制
- 不能初始化原因代码 -不能初始化原因描述
- 检测对象指针 -通过 CheckPointer() 函数返回计算类型指针
- 其他常量 -所有其他常量

© 2000-2025, MetaQuotes Ltd.

# 预定义大量代替值

为简化故障排除并获得MQL5程序运行的信息，有一种特殊的宏，是当时建立编辑的值。使用这些常量的最原始的方法是通过 Print() 函数输出值，如示例中所示。

| 常量 | 描述 |
| --- | --- |
| __CPU_ARCHITECTURE__ | 为之编译EX5文件的架构名称（命令集） |
| __DATE__ | 文件编译日期，无时间（小时，分钟，秒等于0） |
| __DATETIME__ | 文件编译日期和时间 |
| __LINE__ | 源代码中的字符串数量，宏放置的位置 |
| __FILE__ | 当前编辑文件的名称 |
| __PATH__ | 当前正在编译的文件的绝对路径 |
| __FUNCTION__ | 函数名称，宏主体位置 |
| __FUNCSIG__ | 宏所在的函数的签名。函数的完整描述记录可用于识别重载函数很 |
| __MQLBUILD__, __MQL5BUILD__ | 编译器构造数量 |
| __COUNTER__ | 对于每个遇到的__COUNTER__声明，编译器都会将计数器值从0替换到N-1，这里N表示代码中的使用次数。重新编译源代码时，无需做任何更改，即可保证__COUNTER__顺序。<br>__COUNTER__值的计算方法如下：<br>• 初始计数器值为0，<br>• 每次使用计数器后，该值增加1，<br>• 首先，编译器将所有宏和模板现场扩展为源代码，<br>• 为每个模板函数专业化创建单独的代码，<br>• 为每个模板类/结构专业化创建单独的代码，<br>• 接下来，编译器按定义的顺序传递获取的源代码，并用当前计数器值替换每个检测到的__COUNTER__使用。<br>下面的示例显示了编译器如何处理源代码，并用顺序递增的值替换它遇到的__COUNTER__的所有实例。 |
| __RANDOM__ | 编译器为每个__RANDOM__声明插入一个随机ulong值。 |

示例：

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnInit()
  {
//--- EA交易初始化输出信息示例
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
//--- 设置定时器事件间的间隔
   EventSetTimer(5);
//---
  }
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- EA交易无法初始化输出信息示例
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
//---
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 订单收据输出信息
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
   test1(__FUNCTION__);
   test2();
//---
  }
//+------------------------------------------------------------------+
//| test1                                                            |
//+------------------------------------------------------------------+
void test1(string par)
  {
//--- 函数内部信息输出
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__," par=",par);
  }
//+------------------------------------------------------------------+
//| test2                                                            |
//+------------------------------------------------------------------+
void test2()
  {
//--- 函数内部信息输出
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
  }
//+------------------------------------------------------------------+
//| OnTimer event handler                                            |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnTimer()
  {
//---
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
   test1(__FUNCTION__);
  }
```

了解如何使用 __COUNTER__ 宏的示例

```
//--- 创建一个宏，以便在日志中快速显示表达式及其值
#define print(expr) Print(#expr,"=",expr)

//--- 通过预定义的__COUNTER__宏定义MACRO_COUNTER自定义宏
#define MACRO_COUNTER __COUNTER__

//--- 使用__COUNTER__宏设置变量的输入值
input int InpVariable = __COUNTER__;

//--- 定义函数之前，使用__COUNTER__宏设置全局变量的值
int ExtVariable = __COUNTER__;

//+------------------------------------------------------------------+
//| 函数返回__COUNTER__值                                            |
//+------------------------------------------------------------------+
int GlobalFunc(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| 模板函数返回__COUNTER__值                                        |
//+------------------------------------------------------------------+
template<typename T>
int GlobalTemplateFunc(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| 方法返回__COUNTER__的结构                                        |
//+------------------------------------------------------------------+
struct A
  {
   int               dummy;  // not used

   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 方法返回__COUNTER__的模板结构                                    |
//+------------------------------------------------------------------+
template<typename T>
struct B
  {
   int               dummy;  // not used

   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
//+------------------------------------------------------------------+
//| 模板方法返回__COUNTER__的结构                                    |
//+------------------------------------------------------------------+
struct C
  {
   int               dummy;  // not used

   template<typename T>
   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
//+------------------------------------------------------------------+
//| 函数#2，返回__COUNTER__值                                        |
//+------------------------------------------------------------------+
int GlobalFunc2(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
// 宏和变量中的__COUNTER__
   print(MACRO_COUNTER);
   print(InpVariable);
   print(ExtVariable);

//--- 函数中的__COUNTER__
   print(GlobalFunc());
   print(GlobalFunc());                 // 该值没有更改
   print(GlobalTemplateFunc<int>());
   print(GlobalTemplateFunc<int>());    // the value is not changed
   print(GlobalTemplateFunc<double>());// 该值已更改
```

© 2000-2025, MetaQuotes Ltd.

```
   print(GlobalFunc2());
   print(GlobalFunc2());                  // 该值没有更改

// 结构中的__COUNTER__
   A a1, a2;
   print(a1.Method());
   print(a2.Method());                    // 该值没有更改

// 模板结构中的__COUNTER__
   B<int> b1, b2;
   B<double> b3;
   print(b1.Method());
   print(b2.Method());                    // 该值没有更改
   print(b3.Method());                    // 该值已更改

// 模板函数结构中的__COUNTER__
   C c1, c2;
   print(c1.Method<int>());
   print(c1.Method<double>());            // 该值已更改
   print(c2.Method<int>());               // 与第一个c1.Method<int>()调用中的相同值

//--- 让我们再看一下宏和全局变量中的__COUNTER__
   print(MACRO_COUNTER);  //该值已更改
   print(ExtGlobal2);
  }
//--- 定义函数之后，使用__COUNTER__宏设置全局变量的值
int ExtGlobal2 = __COUNTER__;
//+------------------------------------------------------------------+

/* Result
   __COUNTER__=3
   InpVariable=0
   ExtVariable=1
   GlobalFunc()=5
   GlobalFunc()=5
   GlobalTemplateFunc<int>()=8
   GlobalTemplateFunc<int>()=8
   GlobalTemplateFunc<double>()=9
   GlobalFunc2()=7
   GlobalFunc2()=7
   a1.Method()=6
   a2.Method()=6
   b1.Method()=10
   b2.Method()=10
   b3.Method()=11
   c1.Method<int>()=12
   c1.Method<double>()=13
   c2.Method<int>()=12
   __COUNTER__=4
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtGlobal2=2

*/
```

© 2000-2025, MetaQuotes Ltd.

# 数学常量

特殊常量包含价值为数学表达式保留，这些常量可以在程序中的任意位置使用，使用 mathematical functions 取代计算值。

| 常量 | 描述 | 值 |
|---|---|---|
| M_E | e | 2.71828182845904523536 |
| M_LOG2E | log2(e) | 1.44269504088896340736 |
| M_LOG10E | log10(e) | 0.434294481903251827651 |
| M_LN2 | ln(2) | 0.693147180559945309417 |
| M_LN10 | ln(10) | 2.30258509299404568402 |
| M_PI | pi | 3.14159265358979323846 |
| M_PI_2 | pi/2 | 1.57079632679489661923 |
| M_PI_4 | pi/4 | 0.785398163397448309616 |
| M_1_PI | 1/pi | 0.318309886183790671538 |
| M_2_PI | 2/pi | 0.636619772367581343076 |
| M_2_SQRTPI | 2/sqrt(pi) | 1.12837916709551257390 |
| M_SQRT2 | sqrt(2) | 1.41421356237309504880 |
| M_SQRT1_2 | 1/sqrt(2) | 0.70710678118654524401 |

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打印常量值
   Print("M_E = ",DoubleToString(M_E,16));
   Print("M_LOG2E = ",DoubleToString(M_LOG2E,16));
   Print("M_LOG10E = ",DoubleToString(M_LOG10E,16));
   Print("M_LN2 = ",DoubleToString(M_LN2,16));
   Print("M_LN10 = ",DoubleToString(M_LN10,16));
   Print("M_PI = ",DoubleToString(M_PI,16));
   Print("M_PI_2 = ",DoubleToString(M_PI_2,16));
   Print("M_PI_4 = ",DoubleToString(M_PI_4,16));
   Print("M_1_PI = ",DoubleToString(M_1_PI,16));
   Print("M_2_PI = ",DoubleToString(M_2_PI,16));
   Print("M_2_SQRTPI = ",DoubleToString(M_2_SQRTPI,16));
   Print("M_SQRT2 = ",DoubleToString(M_SQRT2,16));
   Print("M_SQRT1_2 = ",DoubleToString(M_SQRT1_2,16));
  }
```

© 2000-2025, MetaQuotes Ltd.

# 数值类型常量

每一简单数值类型都有确定的任务类型，并允许当MQL5程序操作正常使用时最佳化，对于较好的代码可读性和计算值的正确处理，有允许接收设置某一简单数据类型信息的常量。

| 常量 | 描述 | 值 |
|---|---|---|
| CHAR_MIN | 最小值，可以代表图表类型 | -128 |
| CHAR_MAX | 最大值，可以代表图表类型 | 127 |
| UCHAR_MAX | 最大值，可以代表无符字符型 | 255 |
| SHORT_MIN | 最小值，可以代表短型 | -32768 |
| SHORT_MAX | 最大值，可以代表短型 | 32767 |
| USHORT_MAX | 最大值，可以代表无符号短型 | 65535 |
| INT_MIN | 最小值，可以代表整型 | -2147483648 |
| INT_MAX | 最大值，可以代表整型 | 2147483647 |
| UINT_MAX | 最大值，可以代表无符号整型 | 4294967295 |
| LONG_MIN | 最小值，可以代表长型 | -9223372036854775808 |
| LONG_MAX | 最大值，可以代表长型 | 9223372036854775807 |
| ULONG_MAX | 最大值，可以代表无符号长型 | 18446744073709551615 |
| DBL_MIN | 最小正值，可以代表双精度型 | 2.2250738585072014e-308 |
| DBL_MAX | 最大值，可以代表双精度型 | 1.7976931348623158e+308 |
| DBL_EPSILON | 最小值，满足条件1.0+DBL_EPSILON != 1.0 (双精度类型) | 2.2204460492503131e-016 |
| DBL_DIG | 双精度型有效小数位数字 | 15 |
| DBL_MANT_DIG | 双精度型尾数算在内的二进制 | 53 |
| DBL_MAX_10_EXP | 双精度型指数角度最大小数值 | 308 |
| DBL_MAX_EXP | 双精度型指数角度最大二进制值 | 1024 |
| DBL_MIN_10_EXP | 双精度型指数角度最小小数 | (-307) |
| DBL_MIN_EXP | 双精度型指数角度最小二进制值 | (-1021) |
| FLT_MIN | 最小正值，能代表浮点类型 | 1.175494351e-38 |
| FLT_MAX | 最大值，能代表浮点类型 | 3.402823466e+38 |
| FLT_EPSILON | 最小值，满足因素：1.0+DBL_EPSILON != 1.0 (浮点类型) | 1.192092896e-07 |
| FLT_DIG | 浮点型有效角度数位 | 6 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 值 |
| --- | --- | --- |
| FLT_MANT_DIG | 浮点型二进制计算点数 | 24 |
| FLT_MAX_10_EXP | 浮点型指数角度最大小数值 | 38 |
| FLT_MAX_EXP | 浮点型最大二进制值 | 128 |
| FLT_MIN_10_EXP | 浮点型指数角度最小小数值 | -37 |
| FLT_MIN_EXP | 浮点型指数角度最小二进制值 | (-125) |

示例：

```
void OnStart()
  {
//--- 打印常量值
   printf("CHAR_MIN = %d",CHAR_MIN);
   printf("CHAR_MAX = %d",CHAR_MAX);
   printf("UCHAR_MAX = %d",UCHAR_MAX);
   printf("SHORT_MIN = %d",SHORT_MIN);
   printf("SHORT_MAX = %d",SHORT_MAX);
   printf("USHORT_MAX = %d",USHORT_MAX);
   printf("INT_MIN = %d",INT_MIN);
   printf("INT_MAX = %d",INT_MAX);
   printf("UINT_MAX = %u",UINT_MAX);
   printf("LONG_MIN = %I64d",LONG_MIN);
   printf("LONG_MAX = %I64d",LONG_MAX);
   printf("ULONG_MAX = %I64u",ULONG_MAX);
   printf("EMPTY_VALUE = %.16e",EMPTY_VALUE);
   printf("DBL_MIN = %.16e",DBL_MIN);
   printf("DBL_MAX = %.16e",DBL_MAX);
   printf("DBL_EPSILON = %.16e",DBL_EPSILON);
   printf("DBL_DIG = %d",DBL_DIG);
   printf("DBL_MANT_DIG = %d",DBL_MANT_DIG);
   printf("DBL_MAX_10_EXP =  %d",DBL_MAX_10_EXP);
   printf("DBL_MAX_EXP = %d",DBL_MAX_EXP);
   printf("DBL_MIN_10_EXP = %d",DBL_MIN_10_EXP);
   printf("DBL_MIN_EXP = %d",DBL_MIN_EXP);
   printf("FLT_MIN = %.8e",FLT_MIN);
   printf("FLT_MAX = %.8e",FLT_MAX);
   printf("FLT_EPSILON = %.8e",FLT_EPSILON);
/*
   CHAR_MIN = -128
   CHAR_MAX = 127
   UCHAR_MAX = 255
   SHORT_MIN = -32768
   SHORT_MAX = 32767
   USHORT_MAX = 65535
```

© 2000-2025, MetaQuotes Ltd.

```
    INT_MIN = -2147483648
    INT_MAX = 2147483647
    UINT_MAX = 4294967295
    LONG_MIN = -9223372036854775808
    LONG_MAX = 9223372036854775807
    ULONG_MAX = 18446744073709551615
    EMPTY_VALUE = 1.7976931348623157e+308
    DBL_MIN = 2.2250738585072014e-308
    DBL_MAX = 1.7976931348623157e+308
    DBL_EPSILON = 2.2204460492503131e-16
    DBL_DIG = 15
    DBL_MANT_DIG = 53
    DBL_MAX_10_EXP =  308
    DBL_MAX_EXP = 1024
    DBL_MIN_10_EXP = -307
    DBL_MIN_EXP = -1021
    FLT_MIN = 1.17549435e-38
    FLT_MAX = 3.40282347e+38
    FLT_EPSILON = 1.19209290e-07
*/
  }
```

© 2000-2025, MetaQuotes Ltd.

# 不能初始化原因代码

使用 UninitializeReason() 函数返回初始化原因代码，可能值如下：

| 常量 | 值 | 描述 |
|---|---|---|
| REASON_PROGRAM | 0 | 通过调用 ExpertRemove()函数EA交易终止操作 |
| REASON_REMOVE | 1 | 从图表中删除程序 |
| REASON_RECOMPILE | 2 | 程序重新编译 |
| REASON_CHARTCHANGE | 3 | 更改交易品种或图表周期 |
| REASON_CHARTCLOSE | 4 | 关闭图表 |
| REASON_PARAMETERS | 5 | 使用者改变输入参数 |
| REASON_ACCOUNT | 6 | 由于账户设置的改变，会产生激活另一个账户或者重新连接交易服务器的情况。 |
| REASON_TEMPLATE | 7 | 应用新模板 |
| REASON_INITFAILED | 8 | 值代表 OnInit()处理器，返回非零值 |
| REASON_CLOSE | 9 | 程序端已关闭 |

无法初始化原因代码通过预定函数 OnDeinit (整数常量原因)传递参数。

示例：

```
//+------------------------------------------------------------------+
//| 获得文本描述                                                     |
//+------------------------------------------------------------------+
string getUninitReasonText(int reasonCode)
  {
   string text="";
//---
   switch(reasonCode)
     {
      case REASON_ACCOUNT:
         text="Account was changed";break;
      case REASON_CHARTCHANGE:
         text="Symbol or timeframe was changed";break;
      case REASON_CHARTCLOSE:
         text="Chart was closed";break;
      case REASON_PARAMETERS:
         text="Input-parameter was changed";break;
      case REASON_RECOMPILE:
         text="Program "+__FILE__+" was recompiled";break;
      case REASON_REMOVE:
         text="Program "+__FILE__+" was removed from chart";break;
      case REASON_TEMPLATE:
         text="New template was applied to chart";break;
```

© 2000-2025, MetaQuotes Ltd.

```
      default:text="Another reason";
     }
//---
   return text;
  }
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 获得无法初始化原因代码的第一种方法
   Print(__FUNCTION__,"_Uninitalization reason code = ",reason);
//--- 获得无法初始化原因代码的第二种方法
   Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason));
  }
```

© 2000-2025, MetaQuotes Ltd.

# 检测对象指针

CheckPointer() 函数是用来检测对象指针类型的，函数返回 ENUM_POINTER_TYPE 值中一个。

通过 new() 操作创建POINTER_DYNAMIC型对象， delete() operator 只能用于指针。

所有POINTER_AUTOMATIC类型其他指针，表示对象通过MQL5程序环境自动创建，该对象在使用后自动删除。

**ENUM_POINTER_TYPE**

| 常量 | 描述 |
| --- | --- |
| POINTER_INVALID | 错误指针 |
| POINTER_DYNAMIC | new() 操作创建的对象指针 |
| POINTER_AUTOMATIC | 任意类型对象指针自动创建（不使用new()） |

另见

运行错误， 对象删除操作符删除， 检测指针

© 2000-2025, MetaQuotes Ltd.

# 其他常量

CLR_NONE 常量用来概述缺乏的颜色的，它表示指标不会划分 图解对象 或者 图解系列 。这个常量不包括 Web-color 常量列表，当颜色参数必需时，可以应用到任何位置。

INVALID_HANDLE 常数能用来检测处理文件（参见 FileOpen() 和 FileFindFirst() ）。

| 常量 | 描述 | 值 |
|---|---|---|
| CHARTS_MAX | 在终端与开仓图表类似的最大可能值 | 100 |
| clrNONE | 颜色缺乏 | -1 |
| EMPTY_VALUE | 指标缓冲区的空值 | DBL_MAX |
| INVALID_HANDLE | 错误处理 | -1 |
| IS_DEBUG_MODE | MQL5程序操作的调试方式标志 | 调试方式true,否则是false |
| IS_PROFILE_MODE | mq5-program 运作分析模式的标识 | 分析模式下非零，否则为零 |
| NULL | 任意类型零值 | 0 |
| WHOLE_ARRAY | 代表很多条目剩下，直到数组末尾<br><br>如下，处理全部数组 | -1 |
| WRONG_VALUE | 常量可以隐藏 计算 传递到任何 枚举 类型数组 | -1 |

EMPTY_VALUE常量通常符合指标的值，但并不显示在图表中，例如，内置指标标准偏差是一个周期20，在历史中的前19个字节线并没有显示在图表中，如果使用 iStdDev() 函数增添指标处理器并通过CopyBuffer()复制到指标数组里，然后这些值就与EMPTY_VALUE相等。

可以为 自定义指标 选择指定说明自己指标的空值，当指标在图表中不显示时，使用 PlotIndexSetDouble() 函数 PLOT_EMPTY_VALUE 修饰语。

NULL常数可以分派到任意简单类型的变量中，或者分派到对象结构或者分类指针中。NULL为字符串变量分派表示该变量的全部初始化。

当有必要返回 枚举 值时，WRONG_VALUE常量为事件预定，而且还是一个错误值。例如，当我们需要报告返回值是项目中的值。列举一些函数CheckLineStyle(),通过名称为对象返回线类型、指派。如果通过ObjectGetInteger()类型检测结果是true，返回 ENUM_LINE_STYLE 中的值，否则返回WRONG_VALUE。

```
void OnStart()
  {
   if(CheckLineStyle("MyChartObject")==WRONG_VALUE)
      printf("Error line style getting.");
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 为名称指定物件返回线型                                                |
//+------------------------------------------------------------------+
ENUM_LINE_STYLE CheckLineStyle(string name)
  {
   long style;
//---
   if(ObjectGetInteger(0,name,OBJPROP_STYLE,0,style))
      return((ENUM_LINE_STYLE)style);
   else
      return(WRONG_VALUE);
  }
```

WHOLE_ARRAY常量为函数准备，在过程数组中需要制定元素数量：

- ArrayCopy();
- ArrayMinimum();
- ArrayMaximum();
- FileReadArray();
- FileWriteArray().

如果想要列举，指定位置中的所有数组值从头到尾都是需要处理的，你应该指定WHOLE_ARRAY值。

IS_PROFILE_MODE 常量为了在分析模式下收集正确数据允许改变程序操作。分析允许测量单个程序片段的执行时间（通常由函数组成），以及计算这种调用的数量。在分析模式下，可能会禁止Sleep()函数调用来确定执行时间，如下例：

```
//--- Sleep 可以大大影响（改变）分析结果
if(!IS_PROFILE_MODE) Sleep(100); // 在分析模式禁用 Sleep() 调用
```

IS_PROFILE_MODE 常量值是在编译期通过编译器设置，而在传统模式下设置为零。当在分析模式下启动程序时，会执行一个特殊的编译，并且IS_PROFILE_MODE以非零的值替代。

当需要以调试方式对MQL5程序操作稍作改动，使用IS_DEBUG_MODE常量。例如，在终端记录或者增加额外图表对象中，调试方式需要额外的调试信息。

下面的例子创建了一个标签对象并在脚本运行方式中建立了自己的描述和颜色。为了运行MetaEditor中的脚本调试方式，按F5。如果能从浏览窗口中运行脚本，颜色和文本对象标签就会不同。

示例：

```
//+------------------------------------------------------------------+
//|                                              Check_DEBUG_MODE.mq5 |
//|                        Copyright © 2009, MetaQuotes Software Corp. |
//|                                        https://www.metaquotes.net |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="invisible_label";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object",label_name,"not found. Error code = ",GetLastError());
      //--- 创建标签
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- 设置 X 坐标
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- 设置 Y 坐标
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      ResetLastError();
      if(IS_DEBUG_MODE) // 调试方式
        {
         //--- 显示脚本执行模式信息
         ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
         //--- 设置文本颜色成红色
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
            Print("Unable to set the color. Error",GetLastError());
        }
      else               // 操作模式
        {
         ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
         //--- 设置文本颜色为不可见
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
            Print("Unable to set the color. Error ",GetLastError());
        }
      ChartRedraw();
      DebugBreak();     // 如果处于调试模式，则会终止
     }
  }
```

## Crypt 方法

ENUM_CRYPT_METHOD 枚举用于指定数据变换方法，用在 CryptEncode() 和 CryptDecode() 函数。

### ENUM_CRYPT_METHOD

| 常量 | 描述 |
| --- | --- |
| CRYPT_BASE64 | BASE64 |
| CRYPT_AES128 | AES 128位密钥加密（16字节） |
| CRYPT_AES256 | AES 256位密钥加密（32字节） |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 |
| --- | --- |
| CRYPT_DES | DES 56位密钥加密（7字节） |
| CRYPT_HASH_SHA1 | SHA1 HASH caculation |
| CRYPT_HASH_SHA256 | SHA256 HASH caculation |
| CRYPT_HASH_MD5 | MD5 HASH 计算 |
| CRYPT_ARCH_ZIP | ZIP 归档 |

另见

DebugBreak, 执行MQL程序属性, CryptEncode(), CryptDecode()

© 2000-2025, MetaQuotes Ltd.

# 数据结构

MQL5语言提供5种预先结构：

- MqlDateTime 预定给日期和时间；
- MqlParam 当增加指标时使用 IndicatorCreate() 函数发送输入参数；
- MqlRates 试图操纵历史数据，包括价格，成交量和传播的数据信息；
- MqlBookInfo 为了获取市场深度信息；
- MqlTradeRequest 为了增加交易操作交易需求；
- MqlTradeCheckResult 为了在检查已准备的交易请求之前发送它；
- MqlTradeResult 通过OrderSend()函数，获得交易服务器回应交易请求；
- MqlTradeTransaction 包括交易事务的描述；
- MqlTick 为最多需求信息的当前价格设计快速检索。
- 经济日历结构用来获得实时发送至MetaTrader 5平台的经济日历事件的数据。经济日历函数可以在新报告发布之后立即分析宏观经济参数，因为相关的值可以无延迟地直接从源分发。

© 2000-2025, MetaQuotes Ltd.

# Mql日期时间

数据类型结构包括int的8个字段：

```
struct MqlDateTime
  {
   int year;            // 年
   int mon;             // 月
   int day;             // 天
   int hour;            // 小时
   int min;             // 分钟
   int sec;             // 秒
   int day_of_week;     // 一周中的每天（0-周日，1-周一，...，6-周六）
   int day_of_year;     // 一年中的日期号(1月1日 赋予零值)
  };
```

注释

闰年每年天数day_of_year，从三月开始，数字就与非闰年类似。

示例：

```
void OnStart()
  {
//---
   datetime date1=D'2008.03.01';
   datetime date2=D'2009.03.01';

   MqlDateTime str1,str2;
   TimeToStruct(date1,str1);
   TimeToStruct(date2,str2);
   printf("%02d.%02d.%4d, day of year = %d",str1.day,str1.mon,
          str1.year,str1.day_of_year);
   printf("%02d.%02d.%4d, day of year = %d",str2.day,str2.mon,
          str2.year,str2.day_of_year);
  }
/*  Result:
   01.03.2008, day of year = 60
   01.03.2009, day of year = 59
*/
```

另见

TimeToStruct, 结构和类

© 2000-2025, MetaQuotes Ltd.

# 指标输入参量的结构（MqlParam）

当使用 IndicatorCreate() 函数增加技术指标 处理时，MqlParam 参数结构被提供 输入参量 特殊设计。

```
struct MqlParam
  {
   ENUM_DATATYPE     type;                // 输入参量类型，  ENUM_DATATYPE 值
   long              integer_value;       // 存储整数类型域
   double            double_value;        // 存储双精度类型域
   string            string_value;        // 存储字符串类型域
  };
```

一个指标输入的所有参数都以MqlParam类型数组形式传输的，每一元素的类型领域都通过元素传递指定了数据类型。指标值的每一个元素必须在恰当的领域放在第一位（*integer_value*, *double_value* 或者 *string_value*）在类型领域中依靠ENUM_DATATYPE 值指定。

如果IND_CUSTOM值以指标类型第三个传递到IndicatorCreate()函数中，输入参数数组中的第一个元素一定是 ENUM_DATATYPE 项目中TYPE_STRING 值，而*string_value*领域中一定包含自定义指标名称。

© 2000-2025, MetaQuotes Ltd.

# Mql比率

结构存储有关价位、交易量和传递的信息。

```
struct MqlRates
  {
   datetime time;          // 周期开始时间
   double   open;          // 开盘价
   double   high;          // 周期最高价
   double   low;           // 周期最低价
   double   close;         // 收盘价
   long     tick_volume;   // 订单交易量
   int      spread;        // 点差
   long     real_volume;   // 交易量
  };
```

示例：

```
void OnStart()
  {
   MqlRates rates[];
   int copied=CopyRates(NULL,0,0,100,rates);
   if(copied<=0)
      Print("Error copying price data ",GetLastError());
   else Print("Copied ",ArraySize(rates)," bars");
  }
```

另见

复制比率，接入时间序列

© 2000-2025, MetaQuotes Ltd.

# MqlBookInfo

提供市场深度信息。

```
struct MqlBookInfo
  {
   ENUM_BOOK_TYPE    type;          // 从 ENUM_BOOK_TYPE 列举出的订单类型
   double            price;         // 价格
   long              volume;        // 交易量
   double            volume_real;   // 精确度更高的交易量
  };
```

注释

MqlBookInfo 结构是预定义的，因此它不需要任何声明和描述。若要使用该结构，只需声明此类型的变量。

DOM只有在一些交易品种中可以使用。

示例：

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo about ",Symbol());
     }
   else
     {
      Print("Failed to receive DOM for the symbol ",Symbol());
     }
```

另见

MarketBookAdd, MarketBookRelease, MarketBookGet, DOM中交易订单, 数据类型

© 2000-2025, MetaQuotes Ltd.

# 交易请求结构（MqlTradeRequest）

客户端与交易服务器执行其他安置操作相互作用，通过使用交易请求来执行。交易请求由特殊预定MqlTradeRequest类型的结构来体现，包含必要的执行交易订单。该请求执行结果由MqlTradeResult的类型结构来表示。

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;           // 交易操作类型
   ulong                         magic;            // EA交易 ID（幻数）
   ulong                         order;            // 订单号
   string                        symbol;           // 交易的交易品种
   double                        volume;           // 一手需求的交易量
   double                        price;            // 价格
   double                        stoplimit;        // 订单止损限价点位
   double                        sl;               // 订单止损价位点位
   double                        tp;               // 订单盈利价位点位
   ulong                         deviation;        // 需求价格最可能的偏差
   ENUM_ORDER_TYPE               type;             // 订单类型
   ENUM_ORDER_TYPE_FILLING       type_filling;     // 订单执行类型
   ENUM_ORDER_TYPE_TIME          type_time;        // 订单执行时间
   datetime                      expiration;       // 订单终止期（为 ORDER_TIME_SPECIFIED 类型订单）
   string                        comment;          // 订单注释
   ulong                         position;         // 持仓编号
   ulong                         position_by;      // 反向持仓编号
  };
```

字段描述

| 字段 | 描述 |
| --- | --- |
| 操作 | 交易操作类型，可以是 ENUM_TRADE_REQUEST_ACTIONS 项目值中的一个。 |
| 幻数 | EA交易ID，允许操作分析交易命令过程，当发送交易请求时，每个EA交易都能建立自己独特的ID |
| 订单 | 订单命令按钮，用来修改待办订单。 |
| 交易品种 | 交易品种命令，不需要修改命令和关闭放置操作。 |
| 交易量 | 要求命令成交手数，标识每笔订单真实成交量取决于订单执行类型。 |
| 价格 | 价格，必须执行到达命令，交易品种的市场底单，它的执行类型是“市场执行”SYMBOL_TRADE_EXECUTION_MARKET），TRADE_ACTION_DEAL 类型，不要求价格说明。 |
| stoplimit | 价格价值，当价格达到*price*值时（必要条件），Limit安置待办订单，直到待办订单部能安置。 |
| sl | 在不利于价格活动的情况下的止损数值 |
| tp | 在优惠价格活动的情况下目标数值 |
| 偏差 | 最大价格偏差，指定在 points 中 |
| 类型 | 命令类型，可以是 ENUM_ORDER_TYPE项目值中的一个 |

© 2000-2025, MetaQuotes Ltd.

| 字段 | 描述 |
| --- | --- |
| type_filling | 命令执行类型，可以是 ENUM_ORDER_TYPE_FILLING 项目值中的一个 |
| type_time | 命令终结类型，可以是 ENUM_ORDER_TYPE_TIME 项目值中的一个 |
| 到期 | 命令终结期限（ORDER_TIME_SPECIFIED 命令类型） |
| 注释 | 命令注解文本 |

当在锁仓系统更改持仓或平仓时，要确保指明其单号(MqlTradeRequest::position)。而在单边系统，虽然是通过交易品种名称来确定持仓，也可以指明单号。

发送执行命令 交易操作 需要使用 OrderSend() 函数。对于每个交易操作来说都需要指定强制域；也需要填满可选域。有7个可行情况发送交易命令：

请求执行

开仓交易命令在执行请求模式中（交易根据现行价格要求），需要指定如下9种域：

- 执行
- 交易品种
- 交易量
- 价格
- 止损水平
- 盈利水平
- 误差
- 类型
- _filling类型

也有可能指定“magic”和“comment”域值

立即执行

开仓交易命令在立即执行模式中（流动价格交易），要求如下9种域的规格：

- 执行
- 交易品种
- 交易量
- 价格
- 止损水平
- 盈利水平
- 误差
- 类型
- _filling类型

有可能指定“magic”和“comment”域值。

买卖执行

开仓交易命令在买卖执行方式中，要求指定如下5中域：

- 执行
- 交易品种

© 2000-2025, MetaQuotes Ltd.

- 交易量
- 类型
- _filling类型

也有可能指定“magic”和“comment”域值。

交易执行

这是一个在交易执行模式下持仓的交易订单。它需要指定以下5个字段：

- 执行
- 交易品种
- 交易量
- 类型
- _filling类型

也有可能指定“magic”和“comment”域值。

打开买入持仓TRADE_ACTION_DEAL交易操作的示例：

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 打开买入持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 请求的参数
   request.action   =TRADE_ACTION_DEAL;                     // 交易操作类型
   request.symbol   =Symbol();                              // 交易品种
   request.volume   =0.1;                                   // 0.1手交易量
   request.type     =ORDER_TYPE_BUY;                        // 订单类型
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // 持仓价格
   request.deviation=5;                                     // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                          // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // 如果不能发送请求，输出错误代
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

打开卖出持仓TRADE_ACTION_DEAL交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 打开卖出持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 请求参数
   request.action   =TRADE_ACTION_DEAL;                       // 交易操作类型
   request.symbol   =Symbol();                                // 交易品种
   request.volume   =0.2;                                     // 0.2 手交易量
   request.type     =ORDER_TYPE_SELL;                         // 订单类型
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_BID); // 持仓价格
   request.deviation=5;                                       // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                            // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // 如果不能发送请求，输出错误代
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

平仓TRADE_ACTION_DEAL交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456   // EA交易的幻数
//+------------------------------------------------------------------+
//| 关闭全部持仓                                                     |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // 持仓数
//--- 重做所有持仓
   for(int i=total-1; i>=0; i--)
     {
      //--- 订单的参数
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- 输出持仓信息
      PrintFormat("#%I64u %s  %s  %.2f  %s [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  magic);
      //--- 如果幻数匹配
      if(magic==EXPERT_MAGIC)
        {
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置操作参数
         request.action   =TRADE_ACTION_DEAL;        // 交易操作类型
         request.position =position_ticket;          // 持仓价格
         request.symbol   =position_symbol;          // 交易品种
         request.volume   =volume;                   // 持仓交易量
         request.deviation=5;                        // 允许价格偏差
         request.magic    =EXPERT_MAGIC;             // 持仓幻数
         //--- 根据持仓类型设置价格和订单类型
         if(type==POSITION_TYPE_BUY)
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
            request.type =ORDER_TYPE_SELL;
           }
         else
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
            request.type =ORDER_TYPE_BUY;
           }
         //--- 输出关闭信息
         PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToString
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //---
        }
```

© 2000-2025, MetaQuotes Ltd.

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

SL & TP修正

交易命令在StopLoss和/或TakeProfit价格水平上修改，要求指定如下4种域：

- 执行
- 交易品种
- 止损水平
- 盈利水平

更改持仓止损和止赢值的TRADE_ACTION_SLTP交易操作的示例：

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 更改持仓的止损和止赢                                             |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // 持仓数
//--- 重做所有持仓
   for(int i=0; i<total; i++)
     {
      //--- 订单的参数
      ulong  position_ticket=PositionGetTicket(i);// 持仓价格
      string position_symbol=PositionGetString(POSITION_SYMBOL); // 交易品种
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // 小数位数
      ulong  magic=PositionGetInteger(POSITION_MAGIC); // 持仓的幻数
      double volume=PositionGetDouble(POSITION_VOLUME);    // 持仓交易量
      double sl=PositionGetDouble(POSITION_SL);  // 持仓止损
      double tp=PositionGetDouble(POSITION_TP);  // 持仓止赢
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- 输出持仓信息
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- 如果幻数匹配，不定义止损和止赢
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 计算当前价格水平
         double price=PositionGetDouble(POSITION_PRICE_OPEN);
         double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
         double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
         int    stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_L
         double price_level;
         //--- 如果最小值被接受，那么当前平仓价的点数偏距不设置
         if(stop_level<=0)
            stop_level=150; // 设置当前平仓价的150点偏距
         else
            stop_level+=50; // 为了可靠性而设置偏距到(SYMBOL_TRADE_STOPS_LEVEL + 50) 点

         //--- 计算并凑整止损和止赢值
         price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
         if(type==POSITION_TYPE_BUY)
           {
            sl=NormalizeDouble(bid-price_level,digits);
            tp=NormalizeDouble(bid+price_level,digits);
           }
         else
           {
            sl=NormalizeDouble(ask+price_level,digits);
            tp=NormalizeDouble(ask-price_level,digits);
           }
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置操作参数
         request.action  =TRADE_ACTION_SLTP; // 交易操作类型
         request.position=position_ticket;   // 持仓价格
         request.symbol=position_symbol;     // 交易品种
         request.sl      =sl;                // 持仓止损
         request.tp      =tp;                // 持仓止赢
         request.magic=EXPERT_MAGIC;         // 持仓的幻数
         //--- 输出更改信息
         PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToStrin
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

### 待办订单

交易命令安置在待办订单中，要求指定如下11种域：

- 行动
- 交易品种
- 交易量
- 价格
- 限制停止
- 止损水平
- 盈利水平

© 2000-2025, MetaQuotes Ltd.

- 类型
- _filling 类型
- _time 类型
- 期满

也有可能指定 “magic” 和 “comment” 域值。

下挂单 TRADE_ACTION_PENDING 交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456                                    // EA交易的幻数
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;   // 订单类型
//+------------------------------------------------------------------+
//| 下挂单                                                           |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- 下挂单的参数
   request.action   =TRADE_ACTION_PENDING;                                  // 交易操作类型
   request.symbol   =Symbol();                                              // 交易品种
   request.volume   =0.1;                                                   //0.1手交易量
   request.deviation=2;                                                     // 允许价格偏差
   request.magic    =EXPERT_MAGIC;                                          // 订单幻数
   int offset = 50;                                                         // 以点数从当前价
   double price;                                                            // 订单触动价
   double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT);                     // value of poi
   int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);                     // 小数位数（精确
   //--- 检查操作类型
   if(orderType==ORDER_TYPE_BUY_LIMIT)
     {
      request.type     =ORDER_TYPE_BUY_LIMIT;                               // 订单类型
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;             // 持仓价格
      request.price    =NormalizeDouble(price,digits);                      //正常开盘价
     }
   else if(orderType==ORDER_TYPE_SELL_LIMIT)
     {
      request.type     =ORDER_TYPE_SELL_LIMIT;                               // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;              // 持仓价
      request.price    =NormalizeDouble(price,digits);                       // 正常开盘价
     }
   else if(orderType==ORDER_TYPE_BUY_STOP)
     {
      request.type =ORDER_TYPE_BUY_STOP;                                     // 订单类型
      price        =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;      // 开盘价
      request.price=NormalizeDouble(price,digits);                           // 正常开盘价
     }
   else if(orderType==ORDER_TYPE_SELL_STOP)
     {
      request.type     =ORDER_TYPE_SELL_STOP;                                // 订单类型
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point;              // 开盘价
      request.price    =NormalizeDouble(price,digits);                       // 正常开盘价
     }
   else Alert("This example is only for placing pending orders");    // 如果没有挂单被选择
//--- 发送请求
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());                      // 如果不能发送
//--- 操作信息
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

### 修改待办订单

交易命令在待办订单价格中修改，它要求指定如下7种域：

© 2000-2025, MetaQuotes Ltd.

- 行动
- 命令
- 价格
- 止损水平
- 盈利水平
- _time类型
- 期满

更改挂单价格水平的 TRADE_ACTION_MODIFY 交易操作的示例：

© 2000-2025, MetaQuotes Ltd.

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
//+------------------------------------------------------------------+
//| 更改挂单                                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // 已下挂单的总数
//--- 重做所有已下的挂单
   for(int i=0; i<total; i++)
     {
      //--- 订单参数
      ulong  order_ticket=OrderGetTicket(i);                                  // 订单价格
      string order_symbol=Symbol();                                           // 交易品种
      int    digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS);       // 小数位数
      ulong  magic=OrderGetInteger(ORDER_MAGIC);                              // 订单的幻数
      double volume=OrderGetDouble(ORDER_VOLUME_CURRENT);                     // 订单的当前交
      double sl=OrderGetDouble(ORDER_SL);                                     // 订单的当前止
      double tp=OrderGetDouble(ORDER_TP);                                     // 订单的当前止
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE);      // 订单类型
      int offset = 50;                                                        // 以点数从当前
      double price;                                                           // 订单触动价
      double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT);               // 点值
      //--- 输出订单信息
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  order_ticket,
                  order_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- 如果幻数匹配，不定义止损和止赢
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         request.action=TRADE_ACTION_MODIFY;                                  // 交易操作类型
         request.order = OrderGetTicket(i);                                   // 订单价格
         request.symbol    =Symbol();                                         // symbol
         request.deviation=5;                                                 // 允许价格偏差
        //--- 根据持仓类型设置价格水平，订单的止损和止赢
         if(type==ORDER_TYPE_BUY_LIMIT)
           {
            price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开盘
           }
         else if(type==ORDER_TYPE_SELL_LIMIT)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         else if(type==ORDER_TYPE_BUY_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
```

© 2000-2025, MetaQuotes Ltd.

```
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         else if(type==ORDER_TYPE_SELL_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                  // 正常开
           }
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
        }
     }
  }
//+------------------------------------------------------------------+
```

### 删除协办订单

交易命令删除一个待办订单，它要求指定如下2种域

- 行动
- 命令

删除挂单的TRADE_ACTION_REMOVE交易操作的示例：

```
#define EXPERT_MAGIC 123456  // EA交易的幻数
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 删除挂单                                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化交易请求和交易请求结果
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // 下挂单的总数
//--- 重做所有已下的挂单
   for(int i=total-1; i>=0; i--)
     {
      ulong  order_ticket=OrderGetTicket(i);                   // order ticket
      ulong  magic=OrderGetInteger(ORDER_MAGIC);               // 订单的幻数
      //--- 如果幻数匹配
      if(magic==EXPERT_MAGIC)
        {
         //--- 归零请求和结果值
         ZeroMemory(request);
         ZeroMemory(result);
         //--- 设置持仓参数
         request.action=TRADE_ACTION_REMOVE;                   // 交易操作的类型
         request.order = order_ticket;                         // 订单价格
         //--- 发送请求
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // 如果不能发送请求，输出错
         //--- 操作信息
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

另见

结构和类, 交易函数, 订单属性

© 2000-2025, MetaQuotes Ltd.

# 交易要求检查结构结果 (MqlTradeCheckResult)

在向服务器发送 交易操作 要求时，推荐检测。该检测使用 OrderCheck()函数执行，检测要求和MqlTradeCheckResult结构变量类型已通过传递，检测结果会记录该变量。

```
struct MqlTradeCheckResult
  {
   uint          retcode;          // 应答码
   double        balance;          // 交易执行后的余额
   double        equity;           // 交易执行后的净值
   double        profit;           // 浮动利润
   double        margin;           // 保证金规定额
   double        margin_free;      // 自由保证金
   double        margin_level;     // 保证金比例
   string        comment;          // 应答码注释 (描述错误)
  };
```

域描述

| 域 | 描述 |
| --- | --- |
| retcode | 返回代码 |
| balance | 在执行交易操作后账户余额 |
| equity | 在执行交易操作后净值 |
| profit | 在执行交易操作后的浮点利润值 |
| margin | 为执行操作要求的保证金 |
| margin_free | 在执行交易操作后的可用保证金 |
| margin_level | 在执行交易操作后建立保证金水平 |
| comment | 注解文本回应代码，错误描述 |

另见

交易请求结构，当前价格结构，发送订单，订单检测

© 2000-2025, MetaQuotes Ltd.

# 交易请求结果结构 (MqlTradeResult)

交易请求结果，交易服务器返回有关交易要求处理结果的相关数据来作为类型的MqlTradeResult特殊预定义结构。

```
struct MqlTradeResult
  {
   uint     retcode;          // 操作返回代码
   ulong    deal;             // 交易订单号，如果完成的话
   ulong    order;            // 订单号，如果下订单的话
   double   volume;           // 交易交易量，经纪人确认的
   double   price;            // 交易价格，经纪人确认的
   double   bid;              // 当前买入价
   double   ask;              // 当前卖出价
   string   comment;          // 经纪人操作注释 (默认填充操作描述)
   uint     request_id;       // 分派期间通过程序端设置Request ID
   int      retcode_external; // 返回外部交易系统代码
  };
```

域描述

| 域 | 描述 |
|---|---|
| retcode | 交易服务器的返回代码 |
| deal | 交易 标签，如果执行订单，对于 TRADE_ACTION_DEAL 交易操作类型来说是可行的 |
| order | 订单 标签，如果安置标签，对于 TRADE_ACTION_PENDING 交易操作类型来说是可行的 |
| volume | 交易成交量，由经纪人确认，取决于 订单填充类型 |
| price | 订单价格，由经纪人确认，取决于误差域 交易请求 和/或 交易操作 |
| bid | 当前市场买价 (请求报价) |
| ask | 当前市场卖价 (请求报价) |
| comment | 经纪人评论操作 (通过操作描述默认填满) |
| request_id | 发送至交易服务器时通过程序端设置Request ID |
| retcode_external | 通过外部交易系统返回的错误代码。这些错误的使用和类型取决于交易操作发送的交易商和外部交易系统 |

交易操作结果返回到MqlTradeResult类型变量，以第二秒参量 OrderSend() 去执行 交易操作 。

程序端修正了当使用OrdersSend() 和 OrderSendAsync()函数发送至交易服务器时的request_id字段的 requestID。程序端接收来自交易服务器的执行交易的信息，并通过含有以下组件作为参数的OnTradeTransaction()函数提交它们用于处理：

- MqlTradeTransaction结构的交易事务说明；
- OrderSend() 或 OrdersSendAsync() 函数发送的交易请求的说明。Request ID由程序端向交易服务器发送，而该请求本身及其request_id被存储在程序端存储器中；
- 交易请求执行结果为MqlTradeResult结构，request_id字段包含该请求的ID。

© 2000-2025, MetaQuotes Ltd.

OnTradeTransaction()函数接收三个输入参数，但后两个仅适用于有TRADE_TRANSACTION_REQUEST 类型的交易分析。在所有其他情况下，不填写有关交易请求及其执行结果的数据。参数分析的例子可以在交易请求结构找到。

当将交易请求发送至服务器时通过程序端为其设置request_id ，主要介绍用于OrderSendAsync()异步函数工作。该标识符允许将执行的操作与此操作发送到OnTradeTransaction()的结果相关联(OrderSend 或 OrderSendAsync 函数调用)。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 随着处理结果发送交易请求                                         |
//+------------------------------------------------------------------+
bool MyOrderSend(MqlTradeRequest request,MqlTradeResult result)
  {
//--- 重置最后一个错误代码到零
   ResetLastError();
//--- 发送请求
   bool success=OrderSend(request,result);
//--- 如果结果失败-找出为什么
   if(!success)
     {
      int answer=result.retcode;
      Print("TradeLog: Trade request failed. Error = ",GetLastError());
      switch(answer)
        {
         //--- 重新报价
         case 10004:
           {
            Print("TRADE_RETCODE_REQUOTE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- 订单没有被服务器接受
         case 10006:
           {
            Print("TRADE_RETCODE_REJECT");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- 无效价格
         case 10015:
           {
            Print("TRADE_RETCODE_INVALID_PRICE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- 无效 SL and/or TP
         case 10016:
           {
            Print("TRADE_RETCODE_INVALID_STOPS");
            Print("request.sl = ",request.sl," request.tp = ",request.tp);
            Print("result.ask = ",result.ask," result.bid = ",result.bid);
            break;
           }
         //--- 无效交易量
         case 10014:
           {
            Print("TRADE_RETCODE_INVALID_VOLUME");
            Print("request.volume = ",request.volume,"   result.volume = ",
                  result.volume);
            break;
           }
         //--- 无足够的钱进行交易操作
         case 10019:
           {
            Print("TRADE_RETCODE_NO_MONEY");
            Print("request.volume = ",request.volume,"   result.volume = ",
```

© 2000-2025, MetaQuotes Ltd.

```
                  result.volume,"   result.comment = ",result.comment);
               break;
              }
            //--- 一些其他的原因, 输出服务器响应码
            default:
              {
               Print("Other answer = ",answer);
              }
           }
         //--- 通过返回错误值通知服务器请求的不成功结果
         return(false);
        }
//--- 发送命令 () 返回真值 - 复制答案
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 交易事务结构（MqlTradeTransaction）

当在交易账户上执行一些明确活动时，它的状态会发生变化。这种活动包括：

- 使用 OrderSend 和 OrderSendAsync 函数在客户端从任何MQL5应用发送交易请求及其进一步执行；
- 通过程序端图形界面发送交易请求及其进一步执行；
- 在服务器上激活挂单和止损订单；
- 在交易服务器上执行操作。

这些操作的结果会执行以下交易事务：

- 处理交易请求；
- 改变持仓订单；
- 改变订单历史记录；
- 改变交易历史记录；
- 改变持仓。

例如，当发送市场买入订单时，它会被处理，为账户创建相应的买入订单，然后执行订单并从持仓列表中移除，添加到历史订单，而相应的交易也会添加到历史记录并创建新的持仓。所有这些操作都是交易事务。

OnTradeTransaction()的特别句柄在MQL5提供，以获得用于账户的交易。句柄的第一个参数获得描述交易事务的MqlTradeTransaction结构。

```
struct MqlTradeTransaction
  {
   ulong                     deal;             // 交易单
   ulong                     order;            // 订单标签
   string                    symbol;           // 交易品种
   ENUM_TRADE_TRANSACTION_TYPE type;           // 交易事务类型
   ENUM_ORDER_TYPE           order_type;       // 订单类型
   ENUM_ORDER_STATE          order_state;      // 订单状态
   ENUM_DEAL_TYPE            deal_type;        // 成交类型
   ENUM_ORDER_TYPE_TIME      time_type;        // 操作期的订单类型
   datetime                  time_expiration;  // 订单到期时间
   double                    price;            // 价格
   double                    price_trigger;    // 限价止损订单激活价格
   double                    price_sl;         // 止损水平
   double                    price_tp;         // 获利水平
   double                    volume;           // 交易量手数
   ulong                     position;         // 持仓价格
   ulong                     position_by;      // 反向持仓价格
  };
```

字段描述

| 字段 | 描述 |
| --- | --- |
| deal | 交易单。 |
| order | 订单标签。 |
| symbol | 执行交易事务的交易品种的名称。 |

© 2000-2025, MetaQuotes Ltd.

| 字段 | 描述 |
| --- | --- |
| type | 交易事务类型。该值可以是一个ENUM_TRADE_TRANSACTION_TYPE 枚举值。 |
| order_type | 交易订单类型。该值可以是一个ENUM_ORDER_TYPE 枚举值。 |
| order_state | 交易订单状态。该值可以是一个 ENUM_ORDER_STATE 枚举值。 |
| deal_type | 成交类型。该值可以是一个 ENUM_DEAL_TYPE 枚举值。 |
| time_type | 接近期限的订单类型。该值可以是一个ENUM_ORDER_TYPE_TIME枚举值。 |
| time_expiration | 挂单到期期限 (用于 ORDER_TIME_SPECIFIED 和 ORDER_TIME_SPECIFIED_DAY订单类型)。 |
| price | 价格。根据交易事务类型，它可能是订单价，成交价或持仓价。 |
| price_trigger | 限价止损订单，止损（激活）价格 (ORDER_TYPE_BUY_STOP_LIMIT 和 ORDER_TYPE_SELL_STOP_LIMIT)。 |
| price_sl | 止损价。根据交易事务类型，它可能涉及订单，交易或持仓。 |
| price_tp | 获利价。根据交易事务类型，它可能涉及订单，交易或持仓。 |
| volume | 交易量手数。根据交易事务类型，它可能表明订单，交易或持仓的当前交易量。 |

收到的交易分析基本参数就是**type** 字段中指定其类型。例如，如果 TRADE_TRANSACTION_REQUEST 类型的交易（已经收到通过服务器处理交易请求的结果），该结构只有一个字段被完整填写 - **type**。其他字段不分析。这样，我们可以分析两个额外的 **request** 和 **result** 参数提交到 OnTradeTransaction() 句柄，显示如下。

拥有交易操作类型的数据，您可以决定在交易账户上分析订单，持仓和交易的当前状态。记住从程序端发送至服务器的交易请求可以生成多个新的交易事务。程序端优先到达不保证。

MqlTradeTransaction 结构根据交易事务类型以不同的方式填写 (ENUM_TRADE_TRANSACTION_TYPE):

**TRADE_TRANSACTION_ORDER_* and TRADE_TRANSACTION_HISTORY_***

填充MqlTradeTransaction 结构的以下字段，用于有关持仓的相关订单，处理 (TRADE_TRANSACTION_ORDER_ADD, TRADE_TRANSACTION_ORDER_UPDATE 和 TRADE_TRANSACTION_ORDER_DELETE) 和历史订单 (TRADE_TRANSACTION_HISTORY_ADD, TRADE_TRANSACTION_HISTORY_UPDATE, TRADE_TRANSACTION_HISTORY_DELETE):

- order - 订单；
- symbol - 订单交易品种名称；
- type - 交易事务类型；
- order_type - 订单类型；
- orders_state - 订单当前状态；
- time_type - 订单过期类型；
- time_expiration - 订单过期时间 (用于拥有 ORDER_TIME_SPECIFIED 和 ORDER_TIME_SPECIFIED_DAY 过期类型的订单)；
- price - 客户指定的订单价格；

© 2000-2025, MetaQuotes Ltd.

- price_trigger – 限价止损订单止损价格（仅用于 ORDER_TYPE_BUY_STOP_LIMIT 和 ORDER_TYPE_SELL_STOP_LIMIT）；
- price_sl - 止损订单价（已执行，如果订单指明）；
- price_tp - 获利订单价（已执行，如果订单指明）；
- volume - 订单当前交易量（未执行）。初始订单交易量可以使用 HistoryOrders* 函数在历史订单中找到。

**TRADE_TRANSACTION_DEAL_***

填充MqlTradeTransaction结构的以下字段用于交易相关的交易事务，处理（TRADE_TRANSACTION_DEAL_ADD, TRADE_TRANSACTION_DEAL_UPDATE 和 TRADE_TRANSACTION_DEAL_DELETE）：

- deal - 交易单；
- order - 订单标签，基于已执行的交易；
- symbol - 交易品种名称；
- type - 交易事务类型；
- deal_type - 交易类型；
- price - 交易价格；
- price_sl - 止损价格（已执行，如果执行交易的订单指明）；
- price_tp - 获利价（已执行，如果执行交易的订单指明）；
- volume - 交易量手数。

**TRADE_TRANSACTION_POSITION**

填充 MqlTradeTransaction 结构的以下字段用于改变与执行交易（TRADE_TRANSACTION_POSITION）无关联的持仓的相关交易事务：

- symbol - 持仓交易品种名称；
- type - 交易事务类型；
- deal_type - 持仓类型（DEAL_TYPE_BUY 或 DEAL_TYPE_SELL）；
- price - 加权平均持仓价；
- price_sl - 止损价；
- price_tp - 获利价；
- volume - 持仓交易量手数，如果已经改变。

> 持仓改变（添加，改变或关闭），执行交易的结果，不会导致发生 TRADE_TRANSACTION_POSITION 事务。

**TRADE_TRANSACTION_REQUEST**

只填充MqlTradeTransaction结构的一个字段，用于描述已被服务器处理的交易请求的交易事务，并且处理结果已经接收（TRADE_TRANSACTION_REQUEST）：

- type - 交易事务类型；

> 只有 type 字段（交易事务类型）必须分析这种事务。OnTradeTransaction 函数（请求和结果）的第二和第三参数必须分析额外的数据。

例如：

© 2000-2025, MetaQuotes Ltd.

```
input int MagicNumber=1234567;

//--- 启用CTrade 交易类并声明这种类的变量
#include <Trade\Trade.mqh>
CTrade trade;
//--- 安装和删除挂单的标识
bool pending_done=false;
bool pending_deleted=false;
//--- 挂单将被存储在这里
ulong order_ticket;
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 设置MagicNumber 标记我们所有订单
   trade.SetExpertMagicNumber(MagicNumber);
//--- 交易请求将通过OrderSendAsync() 函数以非同步的模式发送
   trade.SetAsyncMode(true);
//--- 初始化变量为零
   order_ticket=0;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家订单函数                                                     |
//+------------------------------------------------------------------+
void OnTick()
  {
//---安装挂单
   if(!pending_done)
     {
      double ask=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
      double buy_stop_price=NormalizeDouble(ask+1000*_Point,(int)SymbolInfoInteger(_Sy
      bool res=trade.BuyStop(0.1,buy_stop_price,_Symbol);
      //--- 如果BuyStop()函数成功执行
      if(res)
        {
         pending_done=true;
         //--- 获取从ctrade发送的请求的结果
         MqlTradeResult trade_result;
         trade.Result(trade_result);
         //--- 获得发送请求的request_id
         uint request_id=trade_result.request_id;
         Print("Request has been sent to set a pending order. Request_ID=",request_id)
         //--- 存储订单 (如果使用非同步模式发送到CTrade，则将为零)
         order_ticket=trade_result.order;
         //--- 全部完成后，提前从OnTick() 处理程序退出
         return;
        }
     }
//--- 删除挂单
   if(!pending_deleted)
      //--- 附加检查
      if(pending_done && (order_ticket!=0))
        {
         //--- 尝试删除挂单
         bool res=trade.OrderDelete(order_ticket);
         Print("OrderDelete=",res);
         //--- 删除请求发送成功
         if(res)
```

© 2000-2025, MetaQuotes Ltd.

```
            {
             pending_deleted=true;
             //--- 得到请求执行结果
             MqlTradeResult trade_result;
             trade.Result(trade_result);
             //--- 获得来自结果的 request ID
             uint request_id=trade_result.request_id;
             //--- 在日志中显示
             Print("The request has been sent to delete a pending order #",order_ticket
                   ". Request_ID=",request_id,
                   "\r\n");
             //--- 修正来自请求结果的订单
             order_ticket=trade_result.order;
            }
         }
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction 函数                                            |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//--- 获取事务类型作为枚举值
   ENUM_TRADE_TRANSACTION_TYPE type=(ENUM_TRADE_TRANSACTION_TYPE)trans.type;
//--- 如果事务是请求处理结果，那么只显示它的名称
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      Print(EnumToString(type));
      //--- 显示已处理的 request 字符串名称
      Print("------------RequestDescription\r\n",RequestDescription(request));
      //--- 显示request 结果描述
      Print("------------ResultDescription\r\n",TradeResultDescription(result));
      //--- 存储用于下一个OnTick()处理的删除的订单
      if(result.order!=0)
        {
         //--- 通过在next OnTick() 调用的标识删除该订单
         order_ticket=result.order;
         Print(" Pending order ticket ",order_ticket,"\r\n");
        }
     }
   else // 显示另一事务类型的完整描述
//--- 在日志中显示已接收事务的描述
      Print("------------TransactionDescription\r\n",TransactionDescription(trans));

//---
  }
//+------------------------------------------------------------------+
//| 返回事务文本描述                                                 |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans)
  {
//---
   string desc=EnumToString(trans.type)+"\r\n";
   desc+="Symbol: "+trans.symbol+"\r\n";
   desc+="Deal ticket: "+(string)trans.deal+"\r\n";
   desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
   desc+="Order ticket: "+(string)trans.order+"\r\n";
   desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
   desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
```

© 2000-2025, MetaQuotes Ltd.

```
   desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
   desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
   desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
   desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
//--- 返回获得的字符串
   return desc;
  }
//+------------------------------------------------------------------+
//| 返回交易请求文本描述                                             |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request)
  {
//---
   string desc=EnumToString(request.action)+"\r\n";
   desc+="Symbol: "+request.symbol+"\r\n";
   desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
   desc+="Order ticket: "+(string)request.order+"\r\n";
   desc+="Order type: "+EnumToString(request.type)+"\r\n";
   desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
   desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
   desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
   desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
   desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
   desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
   desc+="Comment: "+request.comment+"\r\n";
//--- 返回获得的字符串
   return desc;
  }
//+------------------------------------------------------------------+
//| 返回请求处理结果的文本描述                                       |
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result)
  {
//---
   string desc="Retcode "+(string)result.retcode+"\r\n";
   desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
   desc+="Order ticket: "+(string)result.order+"\r\n";
   desc+="Deal ticket: "+(string)result.deal+"\r\n";
   desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
   desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
   desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
   desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
   desc+="Comment: "+result.comment+"\r\n";
//--- 返回获得的字符串
   return desc;
  }
```

另见

交易事务类型, OnTradeTransaction()

© 2000-2025, MetaQuotes Ltd.

# 返回当前价格结构（MqlTick）

该构造是存储交易类型的最新价位，它能最快检索有关当前价格的最需要信息。

```
struct MqlTick
  {
   datetime     time;          // 价格更新的最近时间
   double       bid;           // 当前卖价
   double       ask;           // 当前买价
   double       last;          // 最后交易的价格(Last)
   ulong        volume;        // 当前最后价格的交易量
   long         time_msc;      // 价格最后一次更新的时间，以毫秒计算
   uint         flags;         // 报价标识
   double       volume_real;   // 精确度更高的当前最后价格的交易量
  };
```

通过单一调用 SymbolInfoTick() 函数，MqlTick type 变量允许获得要价，开价，最后价和成交量的值。

无论与之前报价相比是否有变化都会填充每个报价参数。因此，可以找出过去任何时刻的正确价格，无需在报价历史搜索之前的值。例如，即使报价到达期间只有卖价变化，该结构仍然包含其他参数，包括之前的买价，交易量等等。

您可以分析报价标识，找出具体更改了哪个数据：

- TICK_FLAG_BID -更改卖价
- TICK_FLAG_ASK -更改买价
- TICK_FLAG_LAST -更改最终成交价
- TICK_FLAG_VOLUME -更改交易量
- TICK_FLAG_BUY -报价是买入交易结果
- TICK_FLAG_SELL -报价是卖出交易结果

示例：

```
void OnTick()
  {
   MqlTick last_tick;
//---
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      Print(last_tick.time,": Bid = ",last_tick.bid,
            " Ask = ",last_tick.ask,"  Volume = ",last_tick.volume);
     }
   else Print("SymbolInfoTick() failed, error = ",GetLastError());
//---
  }
```

另见

结构和类, CopyTicks(), SymbolInfoTick()

© 2000-2025, MetaQuotes Ltd.

# 经济日历结构

这部分描述了处理可在MetaTrader平台直接使用的经济日历的结构。经济日历是一个即时可用的百科全书，描述了宏观经济指标、指标发布日期以及重要程度。宏观经济指标的相关值会在发布时立即发送至MetaTrader平台，并以标签的形式显示在图表上，使您可以根据国家、货币和重要性直观地跟踪所需的指标。

经济日历函数可以根据自定义重要性标准，从必要的国家/货币对的角度对传入事件进行自动分析。

国家描述通过MqlCalendarCountry结构进行设置。它被用于CalendarCountryById()和CalendarCountries()函数

```
struct MqlCalendarCountry
  {
   ulong                                id;                  // 国家ID (ISO 3166-1)
   string                               name;                // 国家文本名称 在当前程序
   string                               code;                // 国家代码名称(ISO 3166-
   string                               currency;            // 国家货币代码
   string                               currency_symbol;     // 国家货币符号
   string                               url_name;            // 在mql5.com网站URL中使
  };
```

事件描述通过MqlCalendarEvent结构进行设置。它被用于CalendarEventById()、CalendarEventByCountry()和 CalendarEventByCurrency()函数

```
struct MqlCalendarEvent
  {
   ulong                                id;                  // 事件ID
   ENUM_CALENDAR_EVENT_TYPE             type;                // 来自ENUM_CALENDAR_EVE
   ENUM_CALENDAR_EVENT_SECTOR           sector;              // 与事件相关的版块
   ENUM_CALENDAR_EVENT_FREQUENCY        frequency;           //事件频率
   ENUM_CALENDAR_EVENT_TIMEMODE         time_mode;           //事件时间模式
   ulong                                country_id;          // 国家ID
   ENUM_CALENDAR_EVENT_UNIT             unit;                // 经济指标价值的度量单位
   ENUM_CALENDAR_EVENT_IMPORTANCE       importance;          // 事件重要性
   ENUM_CALENDAR_EVENT_MULTIPLIER       multiplier;          // 事件指标值乘数
   uint                                 digits;              // 小数位数
   string                               source_url;          // 事件发布的源URL
   string                               event_code;          // 事件代码
   string                               name;                // 程序端语言中的事件文本名
  };
```

事件值通过MqlCalendarValue结构进行设置。它被用于CalendarValueById(), CalendarValueHistoryByEvent(), CalendarValueHistory(), CalendarValueLastByEvent()和CalendarValueLast()函数

© 2000-2025, MetaQuotes Ltd.

```
struct MqlCalendarValue
  {
   ulong                             id;                // 值ID
   ulong                             event_id;          // 事件ID
   datetime                          time;              // 事件日期和时间
   datetime                          period;            // 事件报告时段
   int                               revision;          // 相对于报告时段的已发布指
   long                              actual_value;      // 如果未设置该值，则返回pp
   long                              prev_value;        // 如果未设置该值，则返回pp
   long                              revised_prev_value;// 如果未设置该值，则返回pp
   long                              forecast_value;    //如果未设置该值，则返回ppm
   ENUM_CALENDAR_EVENT_IMPACT        impact_type;       // 对汇率的潜在影响
  //--- 检查该值的函数
   bool                           HasActualValue(void) const;  // 如果设置了actual_value
   bool                           HasPreviousValue(void) const; // 如果设置了prev_value，
   bool                           HasRevisedValue(void) const;  // 如果设置了revised_prev
   bool                           HasForecastValue(void) const; // 如果设置了forecast_val
  //--- 接收该值的函数
   double                         GetActualValue(void) const;  // 如果未设置该值，则返回ac
   double                         GetPreviousValue(void) const; // 如果未设置该值，则返回pr
   double                         GetRevisedValue(void) const;  // 如果未设置该值，则返回re
   double                         GetForecastValue(void) const; // 如果未设置该值，则返回fo
  };
```

MqlCalendarValue结构提供了检查和设置*actual_value*、*forecast_value*、*prev_value*和*revised_prev_value*字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN (-9223372036854775808)。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

处理日历事件的示例：

```
//--- 用实际值（而非整数）创建结构来存储日历事件
struct AdjustedCalendarValue
  {
   ulong                             id;                // 值ID
   ulong                             event_id;          // 事件ID
   datetime                          time;              // 事件日期和时间
   datetime                          period;            // 事件报告时段
   int                               revision;          // 相对于报告时段的已发布指
   double                            actual_value;      // 实际值
   double                            prev_value;        // 过去值
   double                            revised_prev_value;// 修正后的过去值
   double                            forecast_value;    // 预测值
   ENUM_CALENDAR_EVENT_IMPACT        impact_type;       // 对汇率的潜在影响
```

© 2000-2025, MetaQuotes Ltd.

```
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
//--- 欧盟国家代码(ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- 获得所有欧盟事件值
   MqlCalendarValue values[];
//--- 设置事件发生间隔的边界
   datetime date_from=D'01.01.2021';  // 获取2021年的所有事件
   datetime date_to=0;                // 0 指所有已知事件，包括尚未发生的事件
//--- 请求2021年以来的欧盟事件历史
   if(!CalendarValueHistory(values, date_from, date_to, EU_code))
     {
      PrintFormat("Error! Failed to get events for country_code=%s", EU_code);
      PrintFormat("Error code: %d", GetLastError());
      return;
     }
   else
      PrintFormat("Received event values for country_code=%s: %d",
                  EU_code, ArraySize(values));
//--- 减少输出到 "日志"的数组大小
   if(ArraySize(values)>5)
      ArrayResize(values, 5);
//--- 将事件值原样输出到日志，无需检查或转换为实际值
   Print("Output calendar values as they are");
   ArrayPrint(values);

//--- 检查字段值并转换到实际值
//--- 选项1，检查并获取该值
   AdjustedCalendarValue values_adjusted_1[];
   int total=ArraySize(values);
   ArrayResize(values_adjusted_1, total);
//--- 复制检查并调整后的值
   for(int i=0; i<total; i++)
     {
      values_adjusted_1[i].id=values[i].id;
      values_adjusted_1[i].event_id=values[i].event_id;
      values_adjusted_1[i].time=values[i].time;
      values_adjusted_1[i].period=values[i].period;
      values_adjusted_1[i].revision=values[i].revision;
      values_adjusted_1[i].impact_type=values[i].impact_type;
      //--- 检查值并除以1,000,000
      if(values[i].actual_value==LONG_MIN)
         values_adjusted_1[i].actual_value=double("nan");
      else
```

© 2000-2025, MetaQuotes Ltd.

```
         values_adjusted_1[i].actual_value=values[i].actual_value/1000000.;

      if(values[i].prev_value==LONG_MIN)
         values_adjusted_1[i].prev_value=double("nan");
      else
         values_adjusted_1[i].prev_value=values[i].prev_value/1000000.;

      if(values[i].revised_prev_value==LONG_MIN)
         values_adjusted_1[i].revised_prev_value=double("nan");
      else
         values_adjusted_1[i].revised_prev_value=values[i].revised_prev_value/1000000.

      if(values[i].forecast_value==LONG_MIN)
         values_adjusted_1[i].forecast_value=double("nan");
      else
         values_adjusted_1[i].forecast_value=values[i].forecast_value/1000000.;
     }
   Print("The first method to check and get calendar values");
   ArrayPrint(values_adjusted_1);

//--- 选项2，检查并获取该值
   AdjustedCalendarValue values_adjusted_2[];
   ArrayResize(values_adjusted_2, total);
//--- 复制检查并调整后的值
   for(int i=0; i<total; i++)
     {
      values_adjusted_2[i].id=values[i].id;
      values_adjusted_2[i].event_id=values[i].event_id;
      values_adjusted_2[i].time=values[i].time;
      values_adjusted_2[i].period=values[i].period;
      values_adjusted_2[i].revision=values[i].revision;
      values_adjusted_2[i].impact_type=values[i].impact_type;
      //--- 检查并获取值
      if(values[i].HasActualValue())
         values_adjusted_2[i].actual_value=values[i].GetActualValue();
      else
         values_adjusted_2[i].actual_value=double("nan");

      if(values[i].HasPreviousValue())
         values_adjusted_2[i].prev_value=values[i].GetPreviousValue();
      else
         values_adjusted_2[i].prev_value=double("nan");

      if(values[i].HasRevisedValue())
         values_adjusted_2[i].revised_prev_value=values[i].GetRevisedValue();
      else
         values_adjusted_2[i].revised_prev_value=double("nan");

      if(values[i].HasForecastValue())
```

© 2000-2025, MetaQuotes Ltd.

```
         values_adjusted_2[i].forecast_value=values[i].GetForecastValue();
       else
         values_adjusted_2[i].forecast_value=double("nan");
      }
   Print("The second method to check and get calendar values");
   ArrayPrint(values_adjusted_2);

//--- 选项3，获取值 - 无需检查
   AdjustedCalendarValue values_adjusted_3[];
   ArrayResize(values_adjusted_3, total);
//--- 复制检查并调整后的值
   for(int i=0; i<total; i++)
     {
      values_adjusted_3[i].id=values[i].id;
      values_adjusted_3[i].event_id=values[i].event_id;
      values_adjusted_3[i].time=values[i].time;
      values_adjusted_3[i].period=values[i].period;
      values_adjusted_3[i].revision=values[i].revision;
      values_adjusted_3[i].impact_type=values[i].impact_type;
      //--- 获取值，无需检查
      values_adjusted_3[i].actual_value=values[i].GetActualValue();
      values_adjusted_3[i].prev_value=values[i].GetPreviousValue();
      values_adjusted_3[i].revised_prev_value=values[i].GetRevisedValue();
      values_adjusted_3[i].forecast_value=values[i].GetForecastValue();
     }
   Print("The third method to get calendar values - without checks");
   ArrayPrint(values_adjusted_3);
  }
/*
  我们已接收country_code=EU事件值：1051
  原样输出日历值
         [id] [event_id]              [time]            [period] [revision]        [act
  [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3
  [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0
  [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0 -922337203
  [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0
  [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0
  检查和获取日历值的第一种方法
         [id] [event_id]              [time]            [period] [revision] [actual_va
  [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.2
  [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.1
  [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
  [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.0
  [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.1
  检查和获取日历值的第二种方法
         [id] [event_id]              [time]            [period] [revision] [actual_va
  [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.2
  [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.1
  [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
```

© 2000-2025, MetaQuotes Ltd.

```
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.0
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.1
  获取日历值的第三种方法 - 无需检查
         [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.2
   [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.1
   [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.0
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.1
*/
```

事件频率在MqlCalendarEvent结构中指定。列表中设置了可能的值 **ENUM_CALENDAR_EVENT_FREQUENCY**

| ID | 描述 |
|---|---|
| CALENDAR_FREQUENCY_NONE | 发布频率未设置 |
| CALENDAR_FREQUENCY_WEEK | 每周发布一次 |
| CALENDAR_FREQUENCY_MONTH | 每月发布一次 |
| CALENDAR_FREQUENCY_QUARTER | 每季度发布一次 |
| CALENDAR_FREQUENCY_YEAR | 每年发布一次 |
| CALENDAR_FREQUENCY_DAY | 每天发布一次 |

事件类型在MqlCalendarEvent结构中指定。列表中设置了可能的值**ENUM_CALENDAR_EVENT_TYPE**

| ID | 描述 |
|---|---|
| CALENDAR_TYPE_EVENT | 事件（会议，讲话等） |
| CALENDAR_TYPE_INDICATOR | 指标 |
| CALENDAR_TYPE_HOLIDAY | 假期 |

事件相关的经济版块在MqlCalendarEvent结构中指定。列表中设置了可能的值 **ENUM_CALENDAR_EVENT_SECTOR**

| ID | 描述 |
|---|---|
| CALENDAR_SECTOR_NONE | 版块未设置 |
| CALENDAR_SECTOR_MARKET | 市场，交易所 |
| CALENDAR_SECTOR_GDP | 国内生产总值（GDP） |
| CALENDAR_SECTOR_JOBS | 劳动力市场 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| CALENDAR_SECTOR_PRICES | 价格 |
| CALENDAR_SECTOR_MONEY | 货币 |
| CALENDAR_SECTOR_TRADE | 交易 |
| CALENDAR_SECTOR_GOVERNMENT | 政府 |
| CALENDAR_SECTOR_BUSINESS | 业务 |
| CALENDAR_SECTOR_CONSUMER | 消费 |
| CALENDAR_SECTOR_HOUSING | 房屋 |
| CALENDAR_SECTOR_TAXES | 税收 |
| CALENDAR_SECTOR_HOLIDAYS | 假期 |

事件重要性在MqlCalendarEvent结构中指定。列表中设置了可能的值
**ENUM_CALENDAR_EVENT_IMPORTANCE**

| ID | 描述 |
| --- | --- |
| CALENDAR_IMPORTANCE_NONE | 重要性未设置 |
| CALENDAR_IMPORTANCE_LOW | 低重要性 |
| CALENDAR_IMPORTANCE_MODERATE | 中级重要性 |
| CALENDAR_IMPORTANCE_HIGH | 高级重要性 |

用于显示事件值计量单位类型在MqlCalendarEvent结构中指定。列表中设置了可能的值
**ENUM_CALENDAR_EVENT_UNIT**

| ID | 描述 |
| --- | --- |
| CALENDAR_UNIT_NONE | 计量单位未设置 |
| CALENDAR_UNIT_PERCENT | 百分比 |
| CALENDAR_UNIT_CURRENCY | 国家货币 |
| CALENDAR_UNIT_HOUR | 小时 |
| CALENDAR_UNIT_JOB | 工作 |
| CALENDAR_UNIT_RIG | 钻机设备 |
| CALENDAR_UNIT_USD | 美元 |
| CALENDAR_UNIT_PEOPLE | 人们 |
| CALENDAR_UNIT_MORTGAGE | 抵押贷款 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| CALENDAR_UNIT_VOTE | 投票 |
| CALENDAR_UNIT_BARREL | 原油桶数 |
| CALENDAR_UNIT_CUBICFEET | 立方英尺 |
| CALENDAR_UNIT_POSITION | 非商业净持仓 |
| CALENDAR_UNIT_BUILDING | 建筑 |

在某些情况下，经济参数值需要在MqlCalendarEvent结构中设置乘数。可能的乘数值设置在列表中ENUM_CALENDAR_EVENT_MULTIPLIER

| ID | 描述 |
| --- | --- |
| CALENDAR_MULTIPLIER_NONE | 乘数未设置 |
| CALENDAR_MULTIPLIER_THOUSANDS | 数千 |
| CALENDAR_MULTIPLIER_MILLIONS | 数百万 |
| CALENDAR_MULTIPLIER_BILLIONS | 数十亿 |
| CALENDAR_MULTIPLIER_TRILLIONS | 数万亿 |

事件对本国汇率的潜在影响在MqlCalendarValue结构中显示。列表中设置了可能的值ENUM_CALENDAR_EVENT_IMPACT

| ID | 描述 |
| --- | --- |
| CALENDAR_IMPACT_NA | 影响未设置 |
| CALENDAR_IMPACT_POSITIVE | 积极的影响 |
| CALENDAR_IMPACT_NEGATIVE | 消极的影响 |

事件时间在MqlCalendarEvent结构中指定。列表中设置了可能的值ENUM_CALENDAR_EVENT_TIMEMODE

| ID | 描述 |
| --- | --- |
| CALENDAR_TIMEMODE_DATETIME | 源发布事件的确切时间 |
| CALENDAR_TIMEMODE_DATE | 事件需要一整天 |
| CALENDAR_TIMEMODE_NOTIME | 源不发布事件时间 |
| CALENDAR_TIMEMODE_TENTATIVE | 源发布了的是事件一整天而不是确切时间。时间根据事件发生时指定。 |

© 2000-2025, MetaQuotes Ltd.

另见

经济日历

© 2000-2025, MetaQuotes Ltd.

# 错误和警告代码

该章节包括如下描述：

- 交易服务器返回代码 -通过OrderSend()函数分析交易请求发送结果；
- 编辑器警告 -警告信息代码出现在编辑器中（不是错误）；
- 编译错误 -尝试编译失败后的错误信息代码；
- 运行时间错误 -MQL5程序的属性错误代码可以通过使用 GetLastError()函数获得。

© 2000-2025, MetaQuotes Ltd.

# 交易服务器返回代码

通过使用OrderSend()函数，所有执行交易操作的请求都以MqlTradeRequest交易请求结构发送。函数执行结果代替MqlTradeResult结构，*retcode*域包括交易服务器返回代码，所有代码都在ENUM_TRADE_RETURN_CODES项目中描述。

**ENUM_TRADE_RETURN_CODES**

| 代码 | 常量 | 描述 |
|---|---|---|
| 10004 | TRADE_RETCODE_REQUOTE | 报价请求 |
| 10006 | TRADE_RETCODE_REJECT | 拒绝请求 |
| 10007 | TRADE_RETCODE_CANCEL | 交易者取消请求 |
| 10008 | TRADE_RETCODE_PLACED | 安置命令 |
| 10009 | TRADE_RETCODE_DONE | 完成要求 |
| 10010 | TRADE_RETCODE_DONE_PARTIAL | 请求部分完成 |
| 10011 | TRADE_RETCODE_ERROR | 请求处理错误 |
| 10012 | TRADE_RETCODE_TIMEOUT | 超时取消请求 |
| 10013 | TRADE_RETCODE_INVALID | 无效请求 |
| 10014 | TRADE_RETCODE_INVALID_VOLUME | 请求中无效成交量 |
| 10015 | TRADE_RETCODE_INVALID_PRICE | 请求中的无效价格 |
| 10016 | TRADE_RETCODE_INVALID_STOPS | 请求中的无效访问 |
| 10017 | TRADE_RETCODE_TRADE_DISABLED | 关闭交易 |
| 10018 | TRADE_RETCODE_MARKET_CLOSED | 收市 |
| 10019 | TRADE_RETCODE_NO_MONEY | 没有足够的钱实现请求 |
| 10020 | TRADE_RETCODE_PRICE_CHANGED | 改变价格 |
| 10021 | TRADE_RETCODE_PRICE_OFF | 没有报价处理请求 |
| 10022 | TRADE_RETCODE_INVALID_EXPIRATION | 请求中的无效命令截止日期 |
| 10023 | TRADE_RETCODE_ORDER_CHANGED | 改变命令状态 |
| 10024 | TRADE_RETCODE_TOO_MANY_REQUESTS | 太频繁的请求 |
| 10025 | TRADE_RETCODE_NO_CHANGES | 不改变请求 |
| 10026 | TRADE_RETCODE_SERVER_DISABLES_AT | 服务器无效自动交易 |
| 10027 | TRADE_RETCODE_CLIENT_DISABLES_AT | 客户端无效自动交易 |
| 10028 | TRADE_RETCODE_LOCKED | 处理锁住请求 |
| 10029 | TRADE_RETCODE_FROZEN | 命令或安置冻结 |
| 10030 | TRADE_RETCODE_INVALID_FILL | 无效命令填满字节 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 常量 | 描述 |
| --- | --- | --- |
| 10031 | TRADE_RETCODE_CONNECTION | 与服务器无连接 |
| 10032 | TRADE_RETCODE_ONLY_REAL | 只在流水账允许操作 |
| 10033 | TRADE_RETCODE_LIMIT_ORDERS | 待办订单数量达到限制 |
| 10034 | TRADE_RETCODE_LIMIT_VOLUME | 订单成交量和交易品种位置达到限制 |
| 10035 | TRADE_RETCODE_INVALID_ORDER | 错误或禁止的 订单类型 |
| 10036 | TRADE_RETCODE_POSITION_CLOSED | 指定 POSITION_IDENTIFIER的持仓已经被关闭 |
| 10038 | TRADE_RETCODE_INVALID_CLOSE_VOLUME | 平仓交易量超出当前持仓交易量 |
| 10039 | TRADE_RETCODE_CLOSE_ORDER_EXIST | 指定持仓已存在平仓订单。这可能在锁仓系统工作时出现：<br>• 如果持仓的平仓订单已经存在，试图反向平仓<br>• 如果已有平仓订单和新下单的总交易量超出了当前持仓交易量，试图完全或部分平仓 |
| 10040 | TRADE_RETCODE_LIMIT_POSITIONS | 同时存在于账户中的持仓数量可以通过服务器的设置来限定。达到限定数以后，当再尝试下单时服务器会返回TRADE_RETCODE_LIMIT_POSITIONS错误。限定操作的差异取决于账户持仓类型：<br>• 单边持仓 — 考虑持仓数。达到限定时，平台禁止可能增加持仓数的新下单行为。实际上，平台仅允许下单已有持仓的交易品种。由于执行挂单可能导致当前持仓变化但不会增加持仓数，所以不会考虑当前挂单。<br>• 锁仓持仓 —同时考虑持仓和挂单，因为激活挂单经常意味着一个新的持仓。当限定达到时，平台禁止下可以持仓和挂单的新市场订单。 |
| 10041 | TRADE_RETCODE_REJECT_CANCEL | 挂单激活请求被拒绝，订单被取消 |
| 10042 | TRADE_RETCODE_LONG_ONLY | 因为交易品种(POSITION_TYPE_BUY)设置了"只允许买入持仓"的规则，所以请求被拒绝 |
| 10043 | TRADE_RETCODE_SHORT_ONLY | 因为交易品种 (POSITION_TYPE_SELL)设置了"只允许卖出持仓"的规则，所以请求被拒绝 |
| 10044 | TRADE_RETCODE_CLOSE_ONLY | 因为交易品种设置了"只允许平仓"的规则，所以请求被拒绝 |
| 10045 | TRADE_RETCODE_FIFO_CLOSE | 请求被拒绝，因为已为交易账户设置了"仅允许通过FIFO规则平仓"标识 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 常量 | 描述 |
| --- | --- | --- |
| | | (ACCOUNT_FIFO_CLOSE=true) |
| 10046 | TRADE_RETCODE_HEDGE_PROHIBITED | 请求被拒绝，因为该交易账户设置了“禁用单个交易品种的反向持仓”规则。例如，如果该账户是Buy持仓，那么用户就不能建立Sell持仓或下单卖出挂单。该规则仅适用于使用锁仓账户系统的账户 (ACCOUNT_MARGIN_MODE=ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)。 |

© 2000-2025, MetaQuotes Ltd.

# 编译器警告

编译器警告信息只在情报目的和不出现错误信息的情况下显示。

| 代码 | 描述 |
|---|---|
| 21 | 在时间日期字符串中日期显示不完全记录 |
| 22 | 在时间日期中日期的错误代码，要求<br>年 1970 <= X <= 3000<br>月 0 <X <= 12<br>日 0 <X <= 31/30/28 (29 )... |
| 23 | 在时间日期字符串中时间的错误代码，要求<br>小时 0 <= X <24<br>分钟 0 <= X <60 |
| 24 | 在RGB中的无效颜色，RGB部件中的元素在0到255之间 |
| 25 | 换码顺序中的位置字符<br>Known: \n \r \t \\ \" \' \X \x |
| 26 | 函数局部变量里太大的字符 (>512kb)会减少数字 |
| 29 | 列举项已经定义 (副本)-把项目添加到第一个定义里 |
| 30 | 主要宏 |
| 31 | 变量声明但不使用在任何地方 |
| 32 | 构造函数一定是空型 |
| 33 | 解构函数一定要空型 |
| 34 | 常量不在整数范围内 (X> _UI64_MAX \|\| X <_I64_MIN)，并且转变为双精度型函数 |
| 35 | 太长的HEX—超过16个重要字节 (开出高级半字节形式) |
| 36 | 在HEX字符串"0x"中没有半字节 |
| 37 | 无函数—不执行任何命令 |
| 38 | 使用未初始化变量 |
| 41 | 无主体函数，不被调用 |
| 43 | 在铸字过程中可能丢失的数据。示例：int x = (double) z |
| 44 | 转变常量时精确度缺失 (数据)。示例：int x = M_PI |
| 45 | 在比较操作中两个操作数标志之间的区别。示例：(char) c1> (uchar) c2 |
| 46 | 输入函数遇到的问题—需声明#import或者输入隐蔽函数 |
| 47 | 太大的说明—在可执行文件中，不包括额外字节 |
| 48 | 指标缓冲区的数量比所要求的少 |
| 49 | 在指标中无颜色图解绘图 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 50 | 无图解系列描绘指标 |
| 51 | 在脚本中找不到'OnStart'处理器函数 |
| 52 | 以错误参量定义'OnStart'处理器函数 |
| 53 | 只在脚本中定义'OnStart'函数 |
| 54 | 以错误参量定义'OnInit'函数 |
| 55 | 在脚本中不使用'OnInit'函数 |
| 56 | 以错误参量定义'OnDeinit'函数 |
| 57 | 在脚本中不使用'OnDeinit'函数 |
| 58 | 定义两种'OnCalculate'函数，使用固定价格数组的OnCalculate()。 |
| 59 | 当计算复杂整数常量时检测过量装满 |
| 60 | 或许，变量没有初始化 |
| 61 | 该声明会使不能参考特定行上声明的局部变量 |
| 62 | 该声明会使不能参考特定行上声明的全局变量 |
| 63 | 不能用于 静态分配数组 |
| 64 | 变量声明隐藏 预定义变量 |
| 65 | 表达式的值始终是 true/false |
| 66 | 在数学运算中使用变量或bool类型表达式不安全 |
| 67 | 未定义应用一元负操作符到无符号 ulong 类型的结果 |
| 68 | #property版本 属性指定的版本不接受 市场部分；#property 版本的正确格式 id "XXX.YYY" |
| 69 | 发现空控制语句 |
| 70 | 声明 event处理函数 期间的无效函数返回类型或不正确的参数 |
| 71 | 需要一种类型结构的隐式转换 |
| 72 | 该声明使得不能直接访问指定字符串声明的类成员。只能通过 范围解析操作 :: 进行访问。 |
| 73 | 二进制常量太大，高阶数字将被截断 |
| 74 | 继承类方法的参数有一个不同的const修饰符，衍生函数重载了父函数 |
| 75 | 若 转变逐位运算 有负值或太大的转变值，那么执行结果未定义 |
| 76 | 函数必须 返回一个值 |
| 77 | void 函数返回一个值 |
| 78 | 并非所有控制路径返回一个值 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 79 | 全局比例上不允许表达式 |
| 80 | 检查操作符优先级的可能错误；使用括号来阐明这个优先 |
| 81 | 定义两个 OnCalCulate() 。将使用OHLC 版本 |
| 82 | Struct 没有成员，大小被分配1字节 |
| 83 | 函数的返回值应该被检查 |
| 84 | 编译资源指标进行调试。那会拖慢性能。请重新编译指标提高性能 |
| 85 | 字符串中的大型字符代码必须在0-65535之间 |
| 86 | 字符串中未识别的字符 |
| 87 | 没有定义指标窗口属性 (在主窗口或子窗口设置显示)。属性#property indicator_chart_window被应用 |

© 2000-2025, MetaQuotes Ltd.

# 编译错误

在编辑过程中通过内置编译器，MT5显示了关于系统错误删除的错误信息。这些错误列举在给出的下表中。为了编译可执行的源代码，按**F7**。程序包括的错误直到被淘汰的编辑器定义后才能编辑。

| 代码 | 描述 |
| --- | --- |
| 100 | 文件读数误差 |
| 101 | *. EX5错误文件是用来书写记录的 |
| 103 | 没有足够内存完成编辑 |
| 104 | 编辑器不能识别的空语法 |
| 105 | #include文件中的错误文件名 |
| 106 | #include 包含文件中的访问文件错误 (也许因为文件不存在) |
| 108 | 与#define不匹配的名称 |
| 109 | 预处理器的位置命令 (有效 #include，#define，#property，#import) |
| 110 | 编辑器位置交易品种 |
| 111 | 无法实施的函数 (现行描述，不是主体) |
| 112 | 双引号(")省略 |
| 113 | 尖角括号(<)或双引号(")省略 |
| 114 | 单引号(')省略 |
| 115 | 回尖括号">"省略 |
| 116 | 声明中未标识的类型 |
| 117 | 无返回操作或者在所有分支中未发现返回 |
| 118 | 期望调用参量括号 |
| 119 | 错误书写EX5 |
| 120 | 无效访问数组 |
| 121 | 函数不是缺省型，返回操作会返回值 |
| 122 | 破坏者的特殊声明 |
| 123 | 冒号":"缺失 |
| 124 | 变量已声明 |
| 125 | 标识符变量已声明 |
| 126 | 变量名称太长 (>250字节) |
| 127 | 标识符变量已定义 |
| 128 | 结构未定义 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 129 | 相同名称构造成员已定义 |
| 130 | 没有相关构造成员 |
| 131 | 违背一对分支 |
| 132 | 开括号"("预期 |
| 133 | 不对称花括号 (不是"}") |
| 134 | 难以编译 (太多分支，满溢内部堆积水平) |
| 135 | (打开错误文件阅读) |
| 136 | 没有足够内存下载源文件 |
| 137 | 预期变量 |
| 138 | 参考不能初始化 |
| 140 | 预期分配 (抵达描述) |
| 141 | 开括号"{"预期 |
| 142 | 参数只执行 动态数组 |
| 143 | 使用"void"类型不能接受 |
| 144 | 没有成对的 ")"或者 "]", 例如 "(或"["缺席 |
| 145 | 没有成对的 "(or"[", 如下 ")"or"]"缺席 |
| 146 | 错误数组大小 |
| 147 | 太多参量(>64) |
| 149 | 该符号不是预期的 |
| 150 | 无效使用操作 (无效操作数) |
| 151 | 不允许空型表达式 |
| 152 | 期望操作 |
| 153 | 滥用终止 |
| 154 | 分号";"预期 |
| 155 | 逗号","预期 |
| 156 | 分类类型，而不是结构 |
| 157 | 预期表达式 |
| 158 | HEX中的"non HEX character"或者太长数字 (数字大于511) |
| 159 | 字符串长度超过65534字节 |
| 160 | 函数定义无法接受 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 161 | 不是预期的程序结尾 |
| 162 | 禁止架构的前置声明 |
| 163 | 函数名称已定义并有另一个返回类型 |
| 164 | 函数名称已定义并建立不同参量 |
| 165 | 函数名称已定义并应用 |
| 166 | 调用时函数重载并未发现 |
| 167 | 缺省型返回值函数不能返回值 |
| 168 | 函数不能定义 |
| 170 | 预期价值 |
| 171 | *case* 表达式中只有整数常量是有效的 |
| 172 | *switch*中的*case*值已经使用 |
| 173 | 预期整数 |
| 174 | 预期#import表达式文件名称 |
| 175 | 在总体水平上，表达式不允许使用 |
| 176 | 在前";"省略插入语")" |
| 177 | 预计左标记变量 |
| 178 | 不使用表达式结果 |
| 179 | 说明变量在*case* 中不允许使用 |
| 180 | 从字符串到数字的隐式变换 |
| 181 | 数字的隐式变换到字符串 |
| 182 | 重载函数的不明确调用 (几种重载安置) |
| 183 | 不包含*if*的非法符号*else* |
| 184 | 无需使用*switch*的无效*case*或者*default* |
| 185 | 省略号的不恰当使用 |
| 186 | 初始化序列比初始化常量元素多 |
| 187 | 预期*case*常数 |
| 188 | 要求表达式常数 |
| 189 | 不能改变的常数变量 |
| 190 | 预期回括号或逗号 (说明数组成员) |
| 191 | 枚举标识符已经定义 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 192 | 举例不能访问编辑器(const, extern, static) |
| 193 | 使用不同值声明列举成员 |
| 194 | 相同名称定义变量 |
| 195 | 相同名称定义结构 |
| 196 | 期望列举成员名称 |
| 197 | 预期整数表达式 |
| 198 | 常数表达式除以0 |
| 199 | 函数参量的错误代码 |
| 200 | 通过引用的参量一定是变量 |
| 201 | 通过期望引用传递相同类型的变量 |
| 202 | 恒定变量不能通过不恒定变量传递 |
| 203 | 要求正整数常量 |
| 204 | 访问未保护分类成员失败 |
| 205 | 以另一种方式输入以定义 |
| 208 | 未生成可执行文件 |
| 209 | 指标未发现'OnCalculate'入口点 |
| 210 | 只有在环形使用时能执行连续操作 |
| 211 | 错误访问私人分类成员（关闭） |
| 213 | 未声明的结构方式或分类 |
| 214 | 错误访问私人分类方式（关闭） |
| 216 | 不允许结构复制对象 |
| 218 | 数组转变指标 |
| 219 | 不允许数组在结构或分类中初始化 |
| 220 | 分类构造函数不能有参数 |
| 221 | 分类破坏者不能有参数 |
| 222 | 相同名称的分类方式或结构和参量已声明 |
| 223 | 预期操作数 |
| 224 | 以相同名称分类方式或结构，但使用不同参量（声明！=实现） |
| 225 | 未声明定义函数 |
| 226 | ZeroMemory()不允许用于被保护成员或继承对象 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 227 | 重载函数的歧义调用（几个多载集合参量的准确匹配） |
| 228 | 预期变量名称 |
| 229 | 在此不能声明引用 |
| 230 | 以列举名称已经使用 |
| 232 | 预期的分类或结构 |
| 235 | 不能调用'delete'操作去删除数组 |
| 236 | 预期'while'操作 |
| 237 | 'delete'操作需要指针 |
| 238 | 已经对'switch'有"default' |
| 239 | 句法错误 |
| 240 | 逃脱序列只能在字符串中发生（以'\'开头） |
| 241 | 需要数组—方括号'['，不适用于数组，或者没有数组能参与数组参量传递 |
| 242 | 不能通过初始化序列初始化 |
| 243 | 输入不能定义 |
| 244 | 句法错误的优化程序错误 |
| 245 | 声明太多结构（想要简化程序） |
| 246 | 不允许参数转换 |
| 247 | 错误使用"删除"操作 |
| 248 | 不允许宣布参考指针 |
| 249 | 不允许宣布提及参考 |
| 250 | 不允许宣布指标 |
| 251 | 参量列表中不允许架构宣布 |
| 252 | 铸字过程中的无效操作 |
| 253 | 指标只在分类或架构中声明 |
| 256 | 未定义的标识符 |
| 257 | 可执行代码优化程序错误 |
| 258 | 可执行代码生成错误 |
| 260 | "转换"操作的无效表达式 |
| 261 | 字符串常数溢出，简化程序 |
| 262 | 不能转换列举项目 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 263 | 不能使用数据 "虚拟" (分类或结构成员) |
| 264 | 不能调用分类保护方法 |
| 265 | 拒绝虚拟函数返回不同类型 |
| 266 | 结构不能遗传分类 |
| 267 | 分类不能遗传结构 |
| 268 | 构造函数不能是虚拟的 (不允许使用*virtual*说明符) |
| 269 | 结构不能有虚拟方式 |
| 270 | 函数必须有主体 |
| 271 | 禁止系统函数重载 (终端函数) |
| 272 | *Const* 标识符对函数来说没有分类或结构成员 |
| 274 | 在常数方法中不允许改变分类成员 |
| 276 | 不适合的初始化队列 |
| 277 | 参量丢失的缺省值 (默认参量的特殊声明) |
| 278 | 主要的缺省参量 (声明和执行的不同值) |
| 279 | 不允许为恒定对象调用不恒定方式 |
| 280 | 对象有必要访问成员 (建立无分类或结构指示) |
| 281 | 已声明的结构名称不能在申报中使用 |
| 284 | 未受保护的转换 (关闭继承) |
| 285 | 构造和数组不能以输入变量形式使用 |
| 286 | 对于constructor/destructor来说*Const*无效 |
| 287 | 表示日期时间的错误字符串 |
| 288 | 未知属性 (#property) |
| 289 | 属性错误值 |
| 290 | #property中的性质无效索引 |
| 291 | 调用省略参数-<func (x,)> |
| 293 | 对象通过引用传递 |
| 294 | 数组通过引用传递 |
| 295 | 函数以可输出类型声明 |
| 296 | 函数不能以输出形式声明 |
| 297 | 禁止输出录入函数 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 298 | 导入函数不能有此参量（禁止传递指针，分类或者结构包括动态数据、指针，分类等） |
| 299 | 需要类 |
| 300 | #import 没有关闭 |
| 302 | 类型错配 |
| 303 | 外部变量 已经初始化 |
| 304 | 没有 输出 函数或者 接入点 |
| 305 | 不允许直接调用 构造函数 |
| 306 | 程序没有被声明为常量 |
| 307 | 程序没有被声明为常量 |
| 308 | 不正确的resource文件大小 |
| 309 | 不正确的resource名称 |
| 310 | resource文件打开错误 |
| 311 | resource文件阅读错误 |
| 312 | 未知resource类型 |
| 313 | resource文件的错误路径 |
| 314 | 已经使用指定的 resource 名称 |
| 315 | 类似于宏函数的预期摘要 |
| 316 | 意想不到的宏定义符号 |
| 317 | 宏形式参数的错误 |
| 318 | 无效的宏参数数量 |
| 319 | 太多的宏参数 |
| 320 | 太复杂，简化宏 |
| 321 | 参数 EnumToString() 只能一个枚举 |
| 322 | resource 名称太长 |
| 323 | 不支持的图片格式（只支持24/32位色深的BMP格式） |
| 324 | 数组不能以操作符声明 |
| 325 | 函数只能在 全局 范围被声明 |
| 326 | 不允许在当前范围声明 |
| 327 | 不允许使用局部变量的值初始化动态变量 |
| 328 | 非法声明一个没有默认构造函数 的对象数组 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 329 | 初始化列表 只允许用于 构造函数 |
| 330 | 初始化列表后无定义函数 |
| 331 | 初始化列表 为空 |
| 332 | 不允许构造函数的数组初始化 |
| 333 | 不允许初始化初始化列表 中的父类成员 |
| 334 | 预计的 整型 表达式 |
| 335 | 数组 超过最大值所需的内存 |
| 336 | 结构 超过最大值所需的内存 |
| 337 | 全局 声明的变量超过最大值所需的内存 |
| 338 | 局部变量 超过最大值所需的内存 |
| 339 | 构造函数 未定义 |
| 340 | 图标文件 的无效名称 |
| 341 | 不能在指定路径打开图标文件 |
| 342 | 图标文件 不正确，不是 ICO 格式 |
| 343 | 使用初始化列表重新初始化类/结构构造函数的成员 |
| 344 | 不允许初始化构造函数初始化列表 的 动态 成员 |
| 345 | 不允许初始化全局类/结构的非动态成员 |
| 346 | 类/结构方法的名称匹配早前声明的成员名称 |
| 347 | 类/结构成员的名称匹配早前声明的方法名称 |
| 348 | 虚函数不能声明为静态 |
| 349 | const 修饰符不允许用于静态函数 |
| 350 | 构造函数 或 析构函数 不能是静态的 |
| 351 | 类或结构的非静态成员/方法不能从静态函数访问 |
| 352 | 操作符 关键字后预计会过载运行 (+,-,[],++,— etc.) |
| 353 | 并非所有操作都可以在MQL5 过载 |
| 354 | 定义与声明不匹配 |
| 355 | 为 操作符 指定无效的参数数量 |
| 356 | 未发现Event处理函数 |
| 357 | 方法无法输出 |
| 358 | 常量 对象指针不能通过非常量对象规范化 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 359 | 还不支持类的模板 |
| 360 | 还不支持函数模板 过载 |
| 361 | 不能应用函数模板 |
| 362 | 函数模板的不明确参数（可以应用几种参数类型） |
| 363 | 无法确定规范化函数模板参数的参数类型 |
| 364 | 函数模板中的不正确的参数数量 |
| 365 | 函数模板不能是 虚拟模板 |
| 366 | 函数模板不能导出 |
| 367 | 函数模板不能导入 |
| 368 | 不允许包含对象的结构 |
| 369 | 不允许包含对象的字符串数组和结构 |
| 370 | 静态类/结构成员 必须被显式初始化 |
| 371 | 编译器限制：字符串不能包含超过65 535个字符 |
| 372 | #ifdef/#endif 相悖 |
| 373 | 不能返回类的对象，复制构造函数没有找到 |
| 374 | 不能使用非静态对象和方法 |
| 375 | 没有OnTesterDeinit()不能使用OnTesterInit() |
| 376 | 重新定义正式参数'%s' |
| 377 | 宏 __FUNCSIG__ 和 __FUNCTION__ 不能出现在函数主体之外 |
| 378 | 无效返回类型。例如，从返回结构或指针的DLL输入的函数将会产生这种错误 |
| 379 | 模板使用错误 |
| 380 | 未使用 |
| 381 | 声明纯虚函数时的非法语法，只允许 "=NULL" 或 "=0" |
| 382 | 只有虚函数可以使用纯说明符进行声明 ("=NULL" 或 "=0") |
| 383 | 抽象类不能被实例化 |
| 384 | 用户定义类型指针应该被用作使用dynamic_cast操作符的动态转换目标类型 |
| 385 | 预计的"函数指针"类型 |
| 386 | 不支持类函数指针 |
| 387 | 误差 -不能定义函数指针类型 |
| 388 | 转换类型不可用，由于private 继承 |

© 2000-2025, MetaQuotes Ltd.

| 代码 | 描述 |
| --- | --- |
| 389 | const 修饰符变量应在声明期间初始化 |
| 393 | 只有公共访问的类函数可以界面中声明 |
| 394 | 无效界面 嵌入另一个界面中 |
| 395 | 界面只能由另一个界面衍生 |
| 396 | 期待一个界面 |
| 397 | 界面仅支持公共继承 |
| 398 | 一个界面不能包括成员 |
| 399 | 界面对象不可直接创建，只能通过继承方式 |
| 400 | 说明符不可以用在前向声明中 |
| 401 | 无法从类继承，因为它是通过final说明符声明的 |
| 402 | 无法重定义一个用final说明符声明的方法 |
| 403 | final说明符只能应用于虚函数 |
| 404 | 通过override说明符标记的方法实际上不会覆盖任何基类函数 |
| 405 | 在定义函数时不允许使用说明符，只可以在声明时使用 |
| 406 | 只能将类型转换为指定类型 |
| 407 | 此类型不可以用于resource变量 |
| 408 | 项目文件中的错误 |
| 409 | 无法用作union成员 |
| 410 | 对于名称的模糊选择，应该明确定义使用语境 |
| 411 | 该结构无法从DLL使用 |
| 412 | 无法调用由delete说明符标记的函数 |
| 413 | 不支持MQL4。若要编译程序，请使用MetaTrader 4程序端安装文件夹中的MetaEditor |

© 2000-2025, MetaQuotes Ltd.

# 运行时间错误

GetLastError() 函数是用来返回储存在预定义变量 _LastError中的上一错误代码。该值可以通过 ResetLastError() 函数重置为0

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_SUCCESS | 0 | 操作成功完成 |
| ERR_INTERNAL_ERROR | 4001 | 意外内部错误 |
| ERR_WRONG_INTERNAL_PARAMETER | 4002 | 客户端函数内部调用的错误参数 |
| ERR_INVALID_PARAMETER | 4003 | 当调用系统函数时的错误参量 |
| ERR_NOT_ENOUGH_MEMORY | 4004 | 没有足够空间执行系统函数 |
| ERR_STRUCT_WITHOBJECTS_ORCLASS | 4005 | 结构包括字符串和/或动态数组和/或结构对象和/或分类 |
| ERR_INVALID_ARRAY | 4006 | 数组错误类型，错误大小，或者动态数组的损害对象 |
| ERR_ARRAY_RESIZE_ERROR | 4007 | 数组没有足够空间重置，或者没有能够改变静态数组的大小 |
| ERR_STRING_RESIZE_ERROR | 4008 | 没有足够内存重置字符串 |
| ERR_NOTINITIALIZED_STRING | 4009 | 没有初始化字符串 |
| ERR_INVALID_DATETIME | 4010 | 无效日期和/时间 |
| ERR_ARRAY_BAD_SIZE | 4011 | 数组中的元素总数不能超过 2147483647 |
| ERR_INVALID_POINTER | 4012 | 错误指针 |
| ERR_INVALID_POINTER_TYPE | 4013 | 错误指针类型 |
| ERR_FUNCTION_NOT_ALLOWED | 4014 | 系统函数不允许调用 |
| ERR_RESOURCE_NAME_DUPLICATED | 4015 | 动态 和 静态 resource匹配的名称 |
| ERR_RESOURCE_NOT_FOUND | 4016 | 该名称的resource在EX5没有找到 |
| ERR_RESOURCE_UNSUPPORTED_TYPE | 4017 | 不支持的resource类型或其大小超过 16 Mb |
| ERR_RESOURCE_NAME_IS_TOO_LONG | 4018 | resource名称超过 63 个字符 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_MATH_OVERFLOW | 4019 | 计算数学函数时产生的溢值 |
| ERR_SLEEP_ERROR | 4020 | 调用Sleep()后超过测试截止日期 |
| ERR_PROGRAM_STOPPED | 4022 | 测试从外部强行停止。例如，优化中断、可视化测试窗口关闭或测试代理停止 |
| ERR_INVALID_TYPE | 4023 | 无效类型 |
| ERR_INVALID_HANDLE | 4024 | 无效句柄 |
| ERR_TOO_MANY_OBJECTS | 4025 | 对象池已填写 |
| 图表 | | |
| ERR_CHART_WRONG_ID | 4101 | 错误图表ID |
| ERR_CHART_NO_REPLY | 4102 | 图表不能回应 |
| ERR_CHART_NOT_FOUND | 4103 | 图表未发现 |
| ERR_CHART_NO_EXPERT | 4104 | 图表中没有EA交易可以处理事件 |
| ERR_CHART_CANNOT_OPEN | 4105 | 图表打开错误 |
| ERR_CHART_CANNOT_CHANGE | 4106 | 改变图表交易品种或周期失败 |
| ERR_CHART_WRONG_PARAMETER | 4107 | 图表使用函数 的错误参数值 |
| ERR_CHART_CANNOT_CREATE_TIMER | 4108 | 增加定时器失败 |
| ERR_CHART_WRONG_PROPERTY | 4109 | 错误图表属性ID |
| ERR_CHART_SCREENSHOT_FAILED | 4110 | 错误生成截屏 |
| ERR_CHART_NAVIGATE_FAILED | 4111 | 错误操作图表 |
| ERR_CHART_TEMPLATE_FAILED | 4112 | 错误申请模板 |
| ERR_CHART_WINDOW_NOT_FOUND | 4113 | 未找到窗口包含的指标 |
| ERR_CHART_INDICATOR_CANNOT_ADD | 4114 | 错误添加指标到图表 |
| ERR_CHART_INDICATOR_CANNOT_DEL | 4115 | 删除图表指标的错误 |
| ERR_CHART_INDICATOR_NOT_FOUND | 4116 | 指定图表上未发现指标 |
| 图解对象 | | |
| ERR_OBJECT_ERROR | 4201 | 图解对象的误差 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_OBJECT_NOT_FOUND | 4202 | 未发现图解对象 |
| ERR_OBJECT_WRONG_PROPERTY | 4203 | 图解对象属性的错误ID |
| ERR_OBJECT_GETDATE_FAILED | 4204 | 不能与值相一致获得日期 |
| ERR_OBJECT_GETVALUE_FAILED | 4205 | 不能与日期相一致获得值 |
| 买卖信息 | | |
| ERR_MARKET_UNKNOWN_SYMBOL | 4301 | 未知交易品种 |
| ERR_MARKET_NOT_SELECTED | 4302 | 市场报价中未被挑选出来的交易品种 |
| ERR_MARKET_WRONG_PROPERTY | 4303 | 交易品种属性的错误标识符 |
| ERR_MARKET_LASTTIME_UNKNOWN | 4304 | 时间的最后标志未识别（无标志） |
| ERR_MARKET_SELECT_ERROR | 4305 | 在市场报价中添加或删除交易品种的错误 |
| ERR_MARKET_SELECT_LIMIT | 4306 | 超出了市场报价中选定交易品种的限制 |
| ERR_MARKET_SESSION_INDEX | 4307 | 调用SymbolInfoSessionQuote/SymbolInfoSessionTrade函数时会话ID错误 |
| 历史访问 | | |
| ERR_HISTORY_NOT_FOUND | 4401 | 需求历史未找到 |
| ERR_HISTORY_WRONG_PROPERTY | 4402 | 历史属性的错误ID |
| ERR_HISTORY_TIMEOUT | 4403 | 历史请求超时过多 |
| ERR_HISTORY_BARS_LIMIT | 4404 | 请求柱的数量受到程序端设置的限制 |
| ERR_HISTORY_LOAD_ERRORS | 4405 | 加载历史时发生多个错误 |
| ERR_HISTORY_SMALL_BUFFER | 4407 | 接受数组太小，无法存储所有请求数据 |
| 全局变量 | | |
| ERR_GLOBALVARIABLE_NOT_FOUND | 4501 | 客户端全局变量未找到 |
| ERR_GLOBALVARIABLE_EXISTS | 4502 | 相同名称客户端全局变量已经存在 |
| ERR_GLOBALVARIABLE_NOT_MODIFIED | 4503 | 全局变量未被修改 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_GLOBALVARIABLE_CANNOTREAD | 4504 | 无法通过全局变量值读取文件 |
| ERR_GLOBALVARIABLE_CANNOTWRITE | 4505 | 无法通过全局变量值编写文件 |
| ERR_MAIL_SEND_FAILED | 4510 | 发送邮件失败 |
| ERR_PLAY_SOUND_FAILED | 4511 | 声音播放失败 |
| ERR_MQL5_WRONG_PROPERTY | 4512 | 程序属性错误标识符 |
| ERR_TERMINAL_WRONG_PROPERTY | 4513 | 客户端属性错误标识符 |
| ERR_FTP_SEND_FAILED | 4514 | 通过ftp发送文件失败 |
| ERR_NOTIFICATION_SEND_FAILED | 4515 | 发送 通知 失败 |
| ERR_NOTIFICATION_WRONG_PARAMETER | 4516 | 发送通知的无效参数 - 空字符串或 NULL 被传递到 SendNotification() 函数 |
| ERR_NOTIFICATION_WRONG_SETTINGS | 4517 | 错误的程序端通知设置（未指定ID或未设置权限） |
| ERR_NOTIFICATION_TOO_FREQUENT | 4518 | 发送通知过于频繁 |
| ERR_FTP_NOSERVER | 4519 | 未指定FTP服务器 |
| ERR_FTP_NOLOGIN | 4520 | 未指定FTP登录名 |
| ERR_FTP_FILE_ERROR | 4521 | 在MQL5\Files目录未找到发送至FTP服务器的文件 |
| ERR_FTP_CONNECT_FAILED | 4522 | FTP连接失败 |
| ERR_FTP_CHANGEDIR | 4523 | 在服务器上未找到FTP路径 |
| 自定义指标缓冲区 | | |
| ERR_BUFFERS_NO_MEMORY | 4601 | 没有足够内存建立指标缓冲区 |
| ERR_BUFFERS_WRONG_INDEX | 4602 | 错误指标缓冲区索引 |
| 自定义指标属性 | | |
| ERR_CUSTOM_WRONG_PROPERTY | 4603 | 自定义指标的错误ID |
| 账户 | | |
| ERR_ACCOUNT_WRONG_PROPERTY | 4701 | 错误账户属性ID |
| ERR_TRADE_WRONG_PROPERTY | 4751 | 错误交易属性ID |
| ERR_TRADE_DISABLED | 4752 | EA交易禁止交易 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_TRADE_POSITION_NOT_FOUND | 4753 | 未找到位置 |
| ERR_TRADE_ORDER_NOT_FOUND | 4754 | 未找到命令 |
| ERR_TRADE_DEAL_NOT_FOUND | 4755 | 未找到订单 |
| ERR_TRADE_SEND_FAILED | 4756 | 交易需求发送失败 |
| ERR_TRADE_CALC_FAILED | 4758 | 计算利润或预付款失败 |
| 指标 | | |
| ERR_INDICATOR_UNKNOWN_SYMBOL | 4801 | 未知交易品种 |
| ERR_INDICATOR_CANNOT_CREATE | 4802 | 不能创建指标 |
| ERR_INDICATOR_NO_MEMORY | 4803 | 没有足够内存添加指标 |
| ERR_INDICATOR_CANNOT_APPLY | 4804 | 指标不能适应另一指标 |
| ERR_INDICATOR_CANNOT_ADD | 4805 | 错误申请指标到图表 |
| ERR_INDICATOR_DATA_NOT_FOUND | 4806 | 需求数据未找到 |
| ERR_INDICATOR_WRONG_HANDLE | 4807 | 错误指标处理 |
| ERR_INDICATOR_WRONG_PARAMETERS | 4808 | 当建立指标时使用的错误参量数 |
| ERR_INDICATOR_PARAMETERS_MISSING | 4809 | 当建立指标时没有参量 |
| ERR_INDICATOR_CUSTOM_NAME | 4810 | 在数组中的第一参量一定与自定义指标相同 |
| ERR_INDICATOR_PARAMETER_TYPE | 4811 | 当建立指标时，数组中无效参量类型 |
| ERR_INDICATOR_WRONG_INDEX | 4812 | 错误索引需求的指标缓冲区 |
| 市场深度 | | |
| ERR_BOOKS_CANNOT_ADD | 4901 | 不能添加市场深度 |
| ERR_BOOKS_CANNOT_DELETE | 4902 | 不能移动市场深度 |
| ERR_BOOKS_CANNOT_GET | 4903 | 深度市场数据未获得 |
| ERR_BOOKS_CANNOT_SUBSCRIBE | 4904 | 从市场深度接收新数据时出错 |
| 文件操作 | | |
| ERR_TOO_MANY_FILES | 5001 | 不能同时打开超过64个文件 |
| ERR_WRONG_FILENAME | 5002 | 无效文件名 |
| ERR_TOO_LONG_FILENAME | 5003 | 太长文件名 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_CANNOT_OPEN_FILE | 5004 | 文件打开错误 |
| ERR_FILE_CACHEBUFFER_ERROR | 5005 | 没有足够内存隐藏阅读 |
| ERR_CANNOT_DELETE_FILE | 5006 | 文件删除错误 |
| ERR_INVALID_FILEHANDLE | 5007 | 文件处理关闭，或者根本没打开 |
| ERR_WRONG_FILEHANDLE | 5008 | 错误文件处理 |
| ERR_FILE_NOTTOWRITE | 5009 | 文件必须打开编辑 |
| ERR_FILE_NOTTOREAD | 5010 | 文件必须打开阅读 |
| ERR_FILE_NOTBIN | 5011 | 文件必须以二进制打开 |
| ERR_FILE_NOTTXT | 5012 | 文件必须以文本形式打开 |
| ERR_FILE_NOTTXTORCSV | 5013 | 文件必须以文本或者CSV打开 |
| ERR_FILE_NOTCSV | 5014 | 文件必须以CSV格式打开 |
| ERR_FILE_READERROR | 5015 | 文件阅读错误 |
| ERR_FILE_BINSTRINGSIZE | 5016 | 字符串大小必须制定，因为文件以二进制打开 |
| ERR_INCOMPATIBLE_FILE | 5017 | 文本文件必须以字符串数组打开，或其他数组-二进制 |
| ERR_FILE_IS_DIRECTORY | 5018 | 没有文件，只有目录 |
| ERR_FILE_NOT_EXIST | 5019 | 文件不存在 |
| ERR_FILE_CANNOT_REWRITE | 5020 | 文件不能重写 |
| ERR_WRONG_DIRECTORYNAME | 5021 | 错误目录名称 |
| ERR_DIRECTORY_NOT_EXIST | 5022 | 目录不存在 |
| ERR_FILE_ISNOT_DIRECTORY | 5023 | 只有文件，没有目录 |
| ERR_CANNOT_DELETE_DIRECTORY | 5024 | 目录不能更改 |
| ERR_CANNOT_CLEAN_DIRECTORY | 5025 | 清除目录失败 (或许一个或多个文件阻塞因而操作失败) |
| ERR_FILE_WRITEERROR | 5026 | 将 resource 写入文件失败 |
| ERR_FILE_ENDOFFILE | 5027 | 已到文件末端，不能阅读CSV文件的下一块数据 (FileReadString, FileReadNumber, |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| | | FileReadDatetime, FileReadBool) |
| 字符串分配 | | |
| ERR_NO_STRING_DATE | 5030 | 字符串无日期 |
| ERR_WRONG_STRING_DATE | 5031 | 字符串错误日期 |
| ERR_WRONG_STRING_TIME | 5032 | 字符串错误时间 |
| ERR_STRING_TIME_ERROR | 5033 | 错误转变字符串到日期 |
| ERR_STRING_OUT_OF_MEMORY | 5034 | 没有足够空间建立字符串 |
| ERR_STRING_SMALL_LEN | 5035 | 字符串太短 |
| ERR_STRING_TOO_BIGNUMBER | 5036 | 太多数组，比ULONG_MAX多 |
| ERR_WRONG_FORMATSTRING | 5037 | 无效格式字符串 |
| ERR_TOO_MANY_FORMATTERS | 5038 | 格式说明符数量比参量多 |
| ERR_TOO_MANY_PARAMETERS | 5039 | 参量数多于格式说明符 |
| ERR_WRONG_STRING_PARAMETER | 5040 | 字符串类型破坏参量 |
| ERR_STRINGPOS_OUTOFRANGE | 5041 | 字符串外部位置 |
| ERR_STRING_ZEROADDED | 5042 | 字符串末尾添加0，无用操作 |
| ERR_STRING_UNKNOWNTYPE | 5043 | 当转换字符串时的未知数据类型 |
| ERR_WRONG_STRING_OBJECT | 5044 | 破坏字符串对象 |
| 数组操作 | | |
| ERR_INCOMPATIBLE_ARRAYS | 5050 | 复制不兼容数组，字符串数组只能复制成字符串数组，和数字数组-只在数字数组中 |
| ERR_SMALL_ASSERIES_ARRAY | 5051 | 以AS_SERIES形式接收数据，不够大 |
| ERR_SMALL_ARRAY | 5052 | 太小数组，启动位置在数组外 |
| ERR_ZEROSIZE_ARRAY | 5053 | 0长度数组 |
| ERR_NUMBER_ARRAYS_ONLY | 5054 | 一定是数字数组 |
| ERR_ONEDIM_ARRAYS_ONLY | 5055 | 一定是一维数组 |
| ERR_SERIES_ARRAY | 5056 | 不能使用时序列 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_DOUBLE_ARRAY_ONLY | 5057 | 必须是双精度类型数组 |
| ERR_FLOAT_ARRAY_ONLY | 5058 | 必须是浮点型数组 |
| ERR_LONG_ARRAY_ONLY | 5059 | 必须是长型数组 |
| ERR_INT_ARRAY_ONLY | 5060 | 必须是整型数组 |
| ERR_SHORT_ARRAY_ONLY | 5061 | 必须是短型数组 |
| ERR_CHAR_ARRAY_ONLY | 5062 | 必须是图表型数据 |
| ERR_STRING_ARRAY_ONLY | 5063 | 仅字符串数组 |
| OpenCL操作 | | |
| ERR_OPENCL_NOT_SUPPORTED | 5100 | 该计算机不支持OpenCL函数 |
| ERR_OPENCL_INTERNAL | 5101 | 运行 OpenCL 时发生的内部错误 |
| ERR_OPENCL_INVALID_HANDLE | 5102 | 无效的 OpenCL 句柄 |
| ERR_OPENCL_CONTEXT_CREATE | 5103 | 创建 OpenCL 语境 的错误 |
| ERR_OPENCL_QUEUE_CREATE | 5104 | 在OpenCL创建运行队列失败 |
| ERR_OPENCL_PROGRAM_CREATE | 5105 | 编译 OpenCL 程序 时发生的错误 |
| ERR_OPENCL_TOO_LONG_KERNEL_NAME | 5106 | 过长的内核名称 (OpenCL kernel) |
| ERR_OPENCL_KERNEL_CREATE | 5107 | 创建 OpenCL kernel 的错误 |
| ERR_OPENCL_SET_KERNEL_PARAMETER | 5108 | 当为OpenCL kerne设置参数 时发生的错误 |
| ERR_OPENCL_EXECUTE | 5109 | OpenCL 程序 运行时间错误 |
| ERR_OPENCL_WRONG_BUFFER_SIZE | 5110 | 无效的 OpenCL 缓存大小 |
| ERR_OPENCL_WRONG_BUFFER_OFFSET | 5111 | 无效的 OpenCL 缓存补偿 |
| ERR_OPENCL_BUFFER_CREATE | 5112 | 创建 OpenCL 缓存 失败 |
| ERR_OPENCL_TOO_MANY_OBJECTS | 5113 | OpenCL 对象过多 |
| ERR_OPENCL_SELECTDEVICE | 5114 | OpenCL 设备选择错误 |
| 使用数据库 | | |
| ERR_DATABASE_INTERNAL | 5120 | 内部数据库错误 |
| ERR_DATABASE_INVALID_HANDLE | 5121 | 无效数据库句柄 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_DATABASE_TOO_MANY_OBJECTS | 5122 | 超出数据库对象的最大可接受数量 |
| ERR_DATABASE_CONNECT | 5123 | 数据库连接错误 |
| ERR_DATABASE_EXECUTE | 5124 | 请求执行错误 |
| ERR_DATABASE_PREPARE | 5125 | 请求生成错误 |
| ERR_DATABASE_NO_MORE_DATA | 5126 | 无需再读取数据 |
| ERR_DATABASE_STEP | 5127 | 无法移动到下一个请求条目 |
| ERR_DATABASE_NOT_READY | 5128 | 读取请求结果的数据尚未准备好 |
| ERR_DATABASE_BIND_PARAMETERS | 5129 | 无法自动将参数替换为SQL请求 |
| WebRequest操作 | | |
| ERR_WEBREQUEST_INVALID_ADDRESS | 5200 | 无效URL |
| ERR_WEBREQUEST_CONNECT_FAILED | 5201 | 连接指定URL失败 |
| ERR_WEBREQUEST_TIMEOUT | 5202 | 超时 |
| ERR_WEBREQUEST_REQUEST_FAILED | 5203 | HTTP请求失败 |
| 网络操作（套接） | | |
| ERR_NETSOCKET_INVALIDHANDLE | 5270 | 传递给函数的套接句柄无效 |
| ERR_NETSOCKET_TOO_MANY_OPENED | 5271 | 打开的套接太多（最多128） |
| ERR_NETSOCKET_CANNOT_CONNECT | 5272 | 连接远程主机失败 |
| ERR_NETSOCKET_IO_ERROR | 5273 | 发送数据/从套接接收数据失败 |
| ERR_NETSOCKET_HANDSHAKE_FAILED | 5274 | 建立安全连接失败(TLS Handshake) |
| ERR_NETSOCKET_NO_CERTIFICATE | 5275 | 在保护连接的证书上没有数据 |
| 自定义交易品种 | | |
| ERR_NOT_CUSTOM_SYMBOL | 5300 | 必须指定一个自定义交易品种 |
| ERR_CUSTOM_SYMBOL_WRONG_NAME | 5301 | 自定义交易品种名称无效。交易品种名称只能包含拉丁字母不可含有标点 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| | | 符号，空格或特殊字符（仅可以包含"."，"_"，"&" 和"#"符号）。不建议使用下列字符<，>，:，"，/，\，\|，?，*。 |
| ERR_CUSTOM_SYMBOL_NAME_LONG | 5302 | 自定义交易品种的名称过长。交易品种名称的总长度不可超过32个字符，包括末位的0字符 |
| ERR_CUSTOM_SYMBOL_PATH_LONG | 5303 | 自定义交易品种的路径过长。路径长度不应超过128个字符，包括"Custom \\"，交易品种名称，组分隔符和末位的0字符 |
| ERR_CUSTOM_SYMBOL_EXIST | 5304 | 已存在同名的自定义交易品种 |
| ERR_CUSTOM_SYMBOL_ERROR | 5305 | 创建，删除或更改自定义交易品种时出现错误 |
| ERR_CUSTOM_SYMBOL_SELECTED | 5306 | 您正在试图删除市场报价中选定的自定义交易品种 |
| ERR_CUSTOM_SYMBOL_PROPERTY_WRONG | 5307 | 无效的自定义交易品种属性 |
| ERR_CUSTOM_SYMBOL_PARAMETER_ERROR | 5308 | 设置自定义交易品种属性时的错误参数 |
| ERR_CUSTOM_SYMBOL_PARAMETER_LONG | 5309 | 设置自定义交易品种属性时过长的字符串参数 |
| ERR_CUSTOM_TICKS_WRONG_ORDER | 5310 | 数组中的报价未按时间顺序排列 |
| 经济日历 | | |
| ERR_CALENDAR_MORE_DATA | 5400 | 数组大小不足以接收所有值的描述 |
| ERR_CALENDAR_TIMEOUT | 5401 | 超过请求时限 |
| ERR_CALENDAR_NO_DATA | 5402 | 国家未找到 |
| 使用数据库 | | |
| ERR_DATABASE_ERROR | 5601 | 通用错误 |
| ERR_DATABASE_LOGIC | 5602 | SQLite内部逻辑错误 |
| ERR_DATABASE_PERM | 5603 | 拒绝访问 |
| ERR_DATABASE_ABORT | 5604 | 回调程序请求中止 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_DATABASE_BUSY | 5605 | 数据库文件锁定 |
| ERR_DATABASE_LOCKED | 5606 | 数据库表格锁定 |
| ERR_DATABASE_NOMEM | 5607 | 内存不足，无法完成操作 |
| ERR_DATABASE_READONLY | 5608 | 尝试写入只读数据库 |
| ERR_DATABASE_INTERRUPT | 5609 | sqlite3_interrupt()终止操作 |
| ERR_DATABASE_IOERR | 5610 | 磁盘I/O错误 |
| ERR_DATABASE_CORRUPT | 5611 | 数据库磁盘映像损坏 |
| ERR_DATABASE_NOTFOUND | 5612 | sqlite3_file_control()中的未知操作代码 |
| ERR_DATABASE_FULL | 5613 | 数据库已满，插入失败 |
| ERR_DATABASE_CANTOPEN | 5614 | 无法打开数据库文件 |
| ERR_DATABASE_PROTOCOL | 5615 | 数据库锁定协议错误 |
| ERR_DATABASE_EMPTY | 5616 | 仅供内部使用 |
| ERR_DATABASE_SCHEMA | 5617 | 数据库架构改变 |
| ERR_DATABASE_TOOBIG | 5618 | 字符串或BLOB超出大小限制 |
| ERR_DATABASE_CONSTRAINT | 5619 | 由于违反约束而中止 |
| ERR_DATABASE_MISMATCH | 5620 | 数据类型不匹配 |
| ERR_DATABASE_MISUSE | 5621 | 程序库使用错误 |
| ERR_DATABASE_NOLFS | 5622 | 使用主机上不支持的OS功能 |
| ERR_DATABASE_AUTH | 5623 | 授权失败 |
| ERR_DATABASE_FORMAT | 5624 | 未使用 |
| ERR_DATABASE_RANGE | 5625 | 绑定参数错误，索引不正确 |
| ERR_DATABASE_NOTADB | 5626 | 打开的文件不是数据库文件 |
| 矩阵和向量方法 | | |
| ERR_MATRIX_INTERNAL | 5700 | 矩阵/向量执行子系统的内部错误 |
| ERR_MATRIX_NOT_INITIALIZED | 5701 | 矩阵/向量没有初始化 |
| ERR_MATRIX_INCONSISTENT | 5702 | 运算中矩阵/向量的大小不一致 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 代码 | 描述 |
| --- | --- | --- |
| ERR_MATRIX_INVALID_SIZE | 5703 | 无效的矩阵/向量大小 |
| ERR_MATRIX_INVALID_TYPE | 5704 | 无效的矩阵/向量类型 |
| ERR_MATRIX_FUNC_NOT_ALLOWED | 5705 | 函数不可用于此矩阵/向量 |
| ERR_MATRIX_CONTAINS_NAN | 5706 | 矩阵/向量包括非数值(Nan/Inf) |
| ONNX模式 | | |
| ERR_ONNX_INTERNAL | 5800 | ONNX内部错误 |
| ERR_ONNX_NOT_INITIALIZED | 5801 | ONNX运行时API初始化错误 |
| ERR_ONNX_NOT_SUPPORTED | 5802 | 属性或值不受MQL5支持 |
| ERR_ONNX_RUN_FAILED | 5803 | ONNX运行时API运行错误 |
| ERR_ONNX_INVALID_PARAMETERS_COUNT | 5804 | 传递到OnnxRun的参数数量无效 |
| ERR_ONNX_INVALID_PARAMETER | 5805 | 无效参数值 |
| ERR_ONNX_INVALID_PARAMETER_TYPE | 5806 | 无效参数类型 |
| ERR_ONNX_INVALID_PARAMETER_SIZE | 5807 | 无效参数大小 |
| ERR_ONNX_WRONG_DIMENSION | 5808 | 张量维度未设置或无效 |
| 自定义错误 | | |
| ERR_USER_ERROR_FIRST | 65536 | 用户自定义 错误起始于该代码 |

另见

交易服务器返回代码

© 2000-2025, MetaQuotes Ltd.

# 输入和输出常量

常量：

- 文件开始标签
- 文件属性
- 内部文件位置
- 代码页使用
- 邮箱

© 2000-2025, MetaQuotes Ltd.

# 文件开始标签

文件开始标签指定文件访问方法，标签能如下定义：

| 标识符 | 值 | 描述 |
|---|---|---|
| FILE_READ | 1 | 文件只读，使用 FileOpen() 标签。当打开文件规格需要使用FILE_WRITE和/或FILE_READ |
| FILE_WRITE | 2 | 文件书写开放，使用 FileOpen() 标签。当打开文件规格需要使用FILE_WRITE和/或FILE_READ |
| FILE_BIN | 4 | 二进制阅读/书写模式（不需要转换字符串），使用 FileOpen() 标签 |
| FILE_CSV | 8 | CSV文件（所有元素转换到恰当的字符串类型，统一编码或者ansi，并被分离器分离）使用 FileOpen() 标签 |
| FILE_TXT | 16 | 简单文本文件（与CSV文件相同，但不考虑分离器），使用 FileOpen() 标签 |
| FILE_ANSI | 32 | ANSI类型字符串（一字节交易品种），使用 FileOpen() 标签 |
| FILE_UNICODE | 64 | UNICODE类型字符串（两字节交易品种），使用 FileOpen() 标签 |
| FILE_SHARE_READ | 128 | 多个程序共享阅读，使用 FileOpen() 标签，但是当打开文件时，它不能替代表示 FILE_WRITE和/或FILE_READ 标识的必要性。 |
| FILE_SHARE_WRITE | 256 | 多个程序分享输入，使用 FileOpen() 标签，但是当打开文件时，它不能替代表示 FILE_WRITE和/或FILE_READ 标识的必要性。 |
| FILE_REWRITE | 512 | 使用 FileCopy() 和 FileMove() 函数可以重新输入文件，文件应该在书写时存在或者被打开，否则文件不能打开。 |
| FILE_COMMON | 4096 | 所有客户端在共同文件夹中的文件路径 \Terminal\Common\Files，使用FileIsExist() 函数中的 FileOpen()，FileCopy()，FileMove() 标签。 |

当打开一个文件时，一个或多个标签能够制定。有一个综合标签，通过逻辑OR(|)可以连接标签，并放置在两个举例类型标签中。例如，用CSV格式打开一个文件同时阅读和输入时，会制定结合FILE_READ|FILE_WRITE|FILE_CSV.

示例：

```
int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

当制定阅读和书写标签时，有几种特征：

- 如果指定FILE_READ，想要打开当前文件。如果文件不存在，文件打开失败，不能创建新文件。
- FILE_READ|FILE_WRITE -如果指定文件名称不存在，创建新文件。

© 2000-2025, MetaQuotes Ltd.

- FILE_WRITE -以0值创建文件。

当打开文件规格需要使用FILE_WRITE和/或FILE_READ。

阅读文件定义标签，控制优先。最高标签是 FILE_CSV，然后是FILE_BIN，FILE_TXT 最后。因此，如果同时指定多个标签，(FILE_TXT|FILE_CSV 或者 FILE_TXT|FILE_BIN 或者 FILE_BIN|FILE_CSV)，会使用最高优先权标签。

编码类型定义标签优先，FILE_UNICODE最高优先权，然后是FILE_ANSI，因此，如果指定连接FILE_UNICODE|FILE_ANSI，使用FILE_UNICODE标签

如果FILE_UNICODE 或者 FILE_ANSI都没表明，暗含FILE_UNICODE，如果 FILE_CSV， FILE_BIN，FILE_TXT 都未指定，暗含FILE_CSV。

如果以文本文件类型打开阅读 (FILE_TXT 或者 FILE_CSV)， 文件开始有特殊的两个自己显示0xff,0xfe，编码标签 FILE_UNICODE，指定FILE_ANSI。

另见

文件函数

© 2000-2025, MetaQuotes Ltd.

# 文件属性

FileGetInteger() 函数用于获取文件属性。ENUM_FILE_PROPERTY_INTEGER 枚举的所需属性的标识符会在调用期间传递给它。

**ENUM_FILE_PROPERTY_INTEGER**

| ID | 描述 |
|---|---|
| FILE_EXISTS | 检查是否存在 |
| FILE_CREATE_DATE | 创建日期 |
| FILE_MODIFY_DATE | 最近的更改日期 |
| FILE_ACCESS_DATE | 最后访问文件的日期 |
| FILE_SIZE | 文件的字节大小 |
| FILE_POSITION | 文件指针的位置 |
| FILE_END | 获取文件结束标志 |
| FILE_LINE_END | 获取线型结束标志 |
| FILE_IS_COMMON | 在所有程序端共享的文件夹中打开文件（请见 FILE_COMMON） |
| FILE_IS_TEXT | 文件打开为文本文件（请见 FILE_TXT） |
| FILE_IS_BINARY | 文件打开为二进制文件（请见 FILE_BIN） |
| FILE_IS_CSV | 文件打开为 CSV 文件（请见 FILE_CSV） |
| FILE_IS_ANSI | 文件打开为 ANSI 文件（请见 FILE_ANSI） |
| FILE_IS_READABLE | 打开文件为可读文件（请见 FILE_READ） |
| FILE_IS_WRITABLE | 打开文件为可写入文件（请见 FILE_WRITE） |

FileGetInteger() 函数有两个不同的调用选项。在第一个选项中，用于获取文件属性，指定使用 FileOpen()函数打开文件时获得的句柄。该选项允许获得文件的全部属性。

FileGetInteger() 函数的第二个选项通过文件名称返回文件属性的值。使用该选项，只能获得下面的一般属性：

- FILE_EXISTS -存在带有指定名称的文件
- FILE_CREATE_DATE -带有指定名称的文件的创建日期
- FILE_MODIFY_DATE -带有指定名称的文件的修改日期
- FILE_ACCESS_DATE -带有指定名称的文件的最近访问日期
- FILE_SIZE -带有指定名称的文件的大小

当尝试获取除了上面指定的以外属性时，FileGetInteger()调用的第二个选项将会返回错误。

© 2000-2025, MetaQuotes Ltd.

# 内置文件位置

大多的文件函数和数据读/写操作连接在一起，与此同时，使用 FileSeek() 可以规定文件指针位置到内置文件中，这样将会执行下一项读或写的操作。ENUM_FILE_POSITION表达式包括有效指针位置，下一项操作时可以指定以字节转换。

**ENUM_FILE_POSITION**

| 标识符 | 描述 |
| --- | --- |
| SEEK_SET | 文件起点 |
| SEEK_CUR | 文件指针的当前位置 |
| SEEK_END | 文件末尾 |

另见

文件结束，文件结束线

© 2000-2025, MetaQuotes Ltd.

# 在字符串转换操作使用代码页

当转换字符串变量到字符型数组中时，使用MQL5默认代码相当于ANSI窗口操作系统 (CP_ACP)。如果指定不同类型代码，可以使用 CharArrayToString()， StringToCharArray() 和 FileOpen()。函数建立额外参量。

图表为最受欢迎的代码页列出了内置参量，未提及的代码页通过代码与页相一致来指定。

内置代码页常量

| 常量 | 值 | 描述 |
| --- | --- | --- |
| CP_ACP | 0 | 当前窗口ANSI代码页 |
| CP_OEMCP | 1 | 当前系统OEM代码页 |
| CP_MACCP | 2 | 当前系统Macintosh代码页。<br>注释：值主要在早期创建的程序代码中使用，现在没有用处了，新的Macintosh电脑使用统一码。 |
| CP_THREAD_ACP | 3 | 当前线路使用ANSI窗口代码 |
| CP_SYMBOL | 42 | 交易品种代码页 |
| CP_UTF7 | 65000 | UTF-7代码页 |
| CP_UTF8 | 65001 | UTF-8代码页 |

另见

客户端属性

© 2000-2025, MetaQuotes Ltd.

# 对话框常量

该章包括返回MessageBox()函数代码。万一已经按下ESC键或者取消按钮，如果消息窗口有取消按钮，函数返回IDCANCEL。如果没有取消按钮，按ESC也不起作用。

| 常量 | 值 | 描述 |
|---|---|---|
| IDOK | 1 | 按“确认”键 |
| IDCANCEL | 2 | 按“取消”键 |
| IDABORT | 3 | 按“停止”键 |
| IDRETRY | 4 | 按“重试”键 |
| IDIGNORE | 5 | 按“忽略”键 |
| IDYES | 6 | 按“是”键 |
| IDNO | 7 | 按“否”键 |
| IDTRYAGAIN | 10 | 按“再试一次”键 |
| IDCONTINUE | 11 | 按“继续”键 |

MessageBox() 函数的主要标签定义常量和对话窗口行为，它的值包括综合如下标签组：

| 常量 | 值 | 描述 |
|---|---|---|
| MB_OK | 0x00000000 | 通知窗口只包括一个按钮：确定 默认 |
| MB_OKCANCEL | 0x00000001 | 通知窗口包括两个按钮：确定和取消 |
| MB_ABORTRETRYIGNORE | 0x00000002 | 通知窗口包括三个按钮：停止,重试 和 忽略 |
| MB_YESNOCANCEL | 0x00000003 | 通知窗口包括三个按钮：是，否和取消 |
| MB_YESNO | 0x00000004 | 通知窗口包括两个按钮：是 和 否 |
| MB_RETRYCANCEL | 0x00000005 | 通知窗口包括两个按钮：重试 和 取消 |
| MB_CANCELTRYCONTINUE | 0x00000006 | 通知窗口包括三个按钮：取消，再试一次，继续 |

在消息窗口中展示icon，有必要指定额外标签：

| 常量 | 值 | 描述 |
|---|---|---|
| MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND | 0x00000010 | STOP停止标志icon |
| MB_ICONQUESTION | 0x00000020 | icon疑问标志icon |
| MB_ICONEXCLAMATION, | 0x00000030 | 惊叹/警示标志icon |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 值 | 描述 |
| --- | --- | --- |
| MB_ICONWARNING | | |
| MB_ICONINFORMATION, MB_ICONASTERISK | 0x00000040 | i包围标志 |

如下标签定义默认按钮：

| 常量 | 值 | 描述 |
| --- | --- | --- |
| MB_DEFBUTTON1 | 0x00000000 | 如果另外按钮MB_DEFBUTTON2, MB_DEFBUTTON3, 或者 MB_DEFBUTTON4 不执行，第一按钮MB_DEFBUTTON1 一默认。 |
| MB_DEFBUTTON2 | 0x00000100 | 默认第二按钮 |
| MB_DEFBUTTON3 | 0x00000200 | 默认第三按钮 |
| MB_DEFBUTTON4 | 0x00000300 | 默认第四按钮 |

© 2000-2025, MetaQuotes Ltd.

# MQL5 程序

操作MQL5程序，必须要编译（编译按钮或者F7键），要进行无错误编辑（可以传递警告信息，可供分析），这一过程，同样名称和EX5延期的可执行文件会增加到目录中，term inal_dir\MQL5\Experts, term inal_dir\MQL5\indicators 或者 term inal_dir\MQL5\scripts.这些文件可以执行。

MQL5程序的操作特点可如下形容：

- 程序运行 -调用预先定义的事件处理器的命令；
- 测试交易策略 -策略测试中 MQL5 程序的操作特点。
- 客户端事件 -事件描述，在程序中可以处理；
- 调用输入函数 -描述命令，允许参量，搜索细节并为输入函数同意调用；
- 运行错误 -获得关于运行时间和关键性误差的信息。

EA交易，自定义指标和脚本在导航窗口中附属于公开图表Drag'n'Drop操作

对于停止操作的专家系统来说，通过图表快捷菜单选择Expert Advisors-Rem ove来删除，EA交易的操作也会受到Enable/disable Expert Advisors按钮影响。

为了停止自定义指标，需要从图表中移除。

直到从图表中移除才能运行自定义指标和EA交易；附加在EA交易和指标中的信息在客户端起始位置保存。

在操作完成时脚本执行一次后就自动删除或改变当前图表信息，否则客户端会停止工作，在重启客户端脚本后并不能开始，因为他们的信息并未保存。

EA交易的一个最大化，一个脚本和在一个图表中指标无限的操作数量。

服务不需要绑定图表进行工作，其设计目的在于执行辅助函数。例如，在一项服务中，您可以创建一个自定义交易品种、打开它的图表、使用网络函数在无限循环中接收数据并不断进行更新。

© 2000-2025, MetaQuotes Ltd.

# 程序运行

每个脚本、服务和EA交易都运行在一个单独的线程中。在一个交易品种上计算的所有指标，即使它们附加在不同的图表上，也在同一线程中工作。因此，一个交易品种的所有指标共享一个线程的资源。

与交易品种相关的所有其他操作，例如处理订单号和历史记录同步，也会与指标一起始终在同样的线程中执行。这意味着如果一个指标执行一个无限操作，那么与其交易品种相关的所有其他事件将不会执行。

当运行EA交易时，请确保它有真实的交易环境 和能够访问 所需交易品种的历史记录和周期，并同步程序端和服务器之间的数据。对于所有这些程序，程序端提供了不高于5秒的启动延迟，在这之后，EA交易将和可用的数据一起启动。因此，如果没有连接服务器，这可能导致EA交易的启动延迟。

以下表格包含MQL5程序的简要概述：

| 程序 | 运行 | 注意 |
| --- | --- | --- |
| 服务 | 一个单独的线程，服务的线程数量等于服务的数量 | 循环服务不能中断其他程序的运行 |
| 脚本 | 单独线程，脚本的线程数量等于脚本数 | 循环脚本不能破坏其他程序的运行 |
| EA交易 | 单独线程，EA交易的线程数量等于EA交易的数量。 | 循环EA交易不能破坏其他程序的运行 |
| 指标 | 适用于一个交易品种上的所有指标的一个线程。线程的数量等于指标交易品种的数量 | 一个指标中的无限循环将停止该交易品种上的所有其他指标 |

程序附属于图表之后，上传客户端内存并进行全局变量初始化，如果分类中的某些全局变量有构造函数，在全局变量 初始化时调用构造函数。

在程序等待客户端事件之后，每个MQL5程序应该至少有一个事件处理器，否则就不会执行下载程序，事件处理程序有预定义名称，参量和返回类型。

| 类型 | 函数名称 | 参量 | 应用 | 注释 |
| --- | --- | --- | --- | --- |
| int | OnInit | 无 | EA交易和指标 | 初始化 事件处理器，允许使用缺省返回类型 |
| void | OnDeinit | const int reason | EA交易和指标 | 无法初始化 事件处理器 |
| void | OnStart | 无 | 脚本和服务 | Start (开始)处理程序 |
| int | OnCalculate | const int rates_total, const int prev_calculated, | 指标 | 所有价格计算处理器 |

© 2000-2025, MetaQuotes Ltd.

| 类型 | 函数名称 | 参量 | 应用 | 注释 |
|---|---|---|---|---|
| | | const datetime &Time[],<br>const double &Open[],<br>const double &High[],<br>const double &Low[],<br>const double &Close[],<br>const long &TickVolume[],<br>const long &Volume[],<br>const int &Spread[] | | |
| int | OnCalculate | const int rates_total,<br>const int prev_calculated,<br>const int begin,<br>const double &price[] | 指标 | 计算 单个数据数组事件处理器<br><br>指标不能同时有两个事件处理器<br><br>只有一个事件处理器处理数据数组 |
| void | OnTick | 无 | EA交易 | 新订单事件处理器，处理新接收的文件，该类型没有收到其他文件 |
| void | OnTimer | 无 | EA交易和指标 | 定时器 事件处理器 |
| void | OnTrade | 无 | EA交易 | 交易 事件处理器 |
| double | OnTester | 无 | EA交易 | 测试 事件处理器 |
| void | OnChartEvent | const int id,<br>const long &lparam,<br>const double &dparam,<br>const string &sparam | EA交易和指标 | 图表事件 事件处理器 |
| void | OnBookEvent | const string &symbol_name | EA交易和指标 | 预定事件 事件处理器 |

客户端发送新事件给对应的打开图表。事件也可以通过图表(图表事件)或mql5-程序(自定义事件)生成。生成创建的事件或删除图表上的图形对象可以通过设置CHART_EVENT_OBJECT_CREATE 和 CHART_EVENT_OBJECT_DELETE图表属性来启用或禁用。每个MQL5程序和每个图表都有其自己的事件队列，在这里添加所有新入的事件。

程序仅接收来自它所运行的图表的事件。所有事件按照它们接收的顺序先后处理。如果队列已有NewTick事件，或者该事件当前正被处理，那么新NewTick事件不能置于MQL5程序队列。同样，如果ChartEvent 已经入列，或该事件正被处理，那么此类新事件也不会入列。timer事件以同样的方式处理 - 如果Timer 事件入列或被处理中，新的timer事件就不会入列。

事件队列拥有一个有上限但足够大的容量，这样就不会存在编写良好的程序超出队列大小。如果超出队列容量，新事件就会丢弃，不会入列。

© 2000-2025, MetaQuotes Ltd.

强烈建议不要使用无限循环来处理事件。可能的例外就是处理单独的Start事件的脚本和服务。

程序库 不能处理任何事件。

## 指标和EA交易中禁止的函数

指标，脚本和EA交易是用MQL5编写的可执行程序。它们是专为不同类型的任务而设计的。因此，根据程序类型，在使用某个函数上会有一些限制。以下函数在指标中被禁止：

- OrderCalcMargin();
- OrderCalcProfit();
- OrderCheck();
- OrderSend();
- SendFTP();
- Sleep();
- ExpertRemove();
- MessageBox().

专为指标设计的所有函数在EA交易和脚本中也被禁止：

- SetIndexBuffer();
- IndicatorSetDouble();
- IndicatorSetInteger();
- IndicatorSetString();
- PlotIndexSetDouble();
- PlotIndexSetInteger();
- PlotIndexSetString();
- PlotIndexGetInteger.

程序库不是一个独立的程序，在调用：脚本，指标或EA交易的MQL5程序的情况下被执行。相应地，上述的限制应用到了调用的程序库。

## 服务中禁止的函数

因为服务不绑定图表，进而不接受任何事件。服务中禁止以下函数：

ExpertRemove();

EventSetMillisecondTimer();

EventSetTimer();

EventKillTimer();

SetIndexBuffer();

IndicatorSetDouble();

© 2000-2025, MetaQuotes Ltd.

IndicatorSetInteger();
IndicatorSetString();
PlotIndexSetDouble();
PlotIndexSetInteger();
PlotIndexSetString();
PlotIndexGetInteger();

函数库 不处理任何事件。

## 加载卸载指标

如下步骤装载指标：

- 指标附属于图表；
- 客户端开始（如果指标粘贴在图表之前，那是用来阻止客户端的）；
- 装载样本（在样本中，指标添加到图表指定位置）；
- 剖面转变（指标粘贴在其中一个剖面图里）；
- 图表中的交易品种或者时间表的转变，需要附加指标；
- 改变程序端连接的账号；
- 一个指标重新编译后，指标粘贴在一个图表上；
- 改变指标的 输入参量 。

如下步骤卸载指标：

- 从图表中分类指标；
- 终端关机（指标附属在图表上）；
- 如果指标附加在图表上，下载一个模板；
- 关闭一个图表，指标是附加的；
- 改变一个剖面，改变指标附加的图表中的一个剖面；
- 图表中交易品种或者时间表改变，指标附加；
- 改变程序端连接的账号；
- 改变指标中的 输入参量 。

## EA交易加载卸载

如下步骤下载EA交易：

- 图表上添加EA交易；
- 程序启动（EA交易添加到图表之前，关闭客户端）；
- 下载一个样本（在样本中添加到图表的EA交易是指定的）；
- 改变剖面（EA交易添加到图表中一项剖面图中）；

© 2000-2025, MetaQuotes Ltd.

- 连接账户，即使账号相同（如果在服务器上授权程序端之前，EA交易被附加到图表）。

如下步骤卸载EA交易：

- 从图表中分离EA交易；
- 如果新的EA交易添加到图表中，另一个EA交易已经附加，EA交易就被卸载；
- 关掉客户端（EA交易添加到图表中）；
- 装载模板，EA交易添加到图表中；
- 关闭添加EA交易的图表；
- 剖面转变，EA交易添加到图表中一个剖面图中；
- 改变程序端连接的账号（如果在服务器上授权程序端之前，EA交易被附加到图表）；
- calling the ExpertRemove() function.

**添加EA交易的图表中的交易品种和时间表改变了，EA加以就不能加载或者卸载。**一旦客户端随后以新的交易品种/时间表调用在旧的交易品种/时间表的 OnDeinit() 和 OnInit() 处理器，全局变量的值和 静态变量 不能重新设置。EA交易中接收的所有事件，在初始化之前就完整(OnInit() function)跳过。

## 脚本加载卸载

在添加到图表过后脚本就被迅速装载，在完成操作之后会迅速被卸载，OnInit() 和 OnDeinit() 用来调用脚本文件。

当程序卸载后（从图表中删除），全局变量的客户端程序不能初始化并删除队列中的事件，这样所有的字符类型常量都会重设，动态数组对象的存储单元会分配析构函数也会调用。

## 加载和卸载服务

如果服务在程序端关闭时启动，那么在启动程序端后立即加载这些服务。服务在完成工作后被立即卸载。

服务有一个OnStart()处理程序，在此程序中，您可以实现无休止的数据接收和处理循环，例如使用网络函数创建和更新自定义交易品种。

与EA交易、指标和脚本不同，服务并不绑定特定的图表，因此为其启动提供一个单独的机制。使用“添加服务”命令，在导航中创建一个新服务实例。可以使用对应的实例菜单启动、停止和移除服务实例。要管理所有实例，请使用服务菜单。

为了更好地理解EA交易的操作，我们推荐如下的EA交易编译代码，并执行装载/卸载，模板转变，样本转变，事件表转变等等：

**示例：**

```
//+------------------------------------------------------------------+
//|                                                  TestExpert.mq5 |
```

© 2000-2025, MetaQuotes Ltd.

```
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
  {
public:
   CTestClass() { Print("CTestClass constructor"); }
   ~CTestClass() { Print("CTestClass destructor"); }
  };
CTestClass global;
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   Print("Initialisation");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print("Deinitialisation with reason",reason);
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

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

另见

客户端事件，事件处理程序

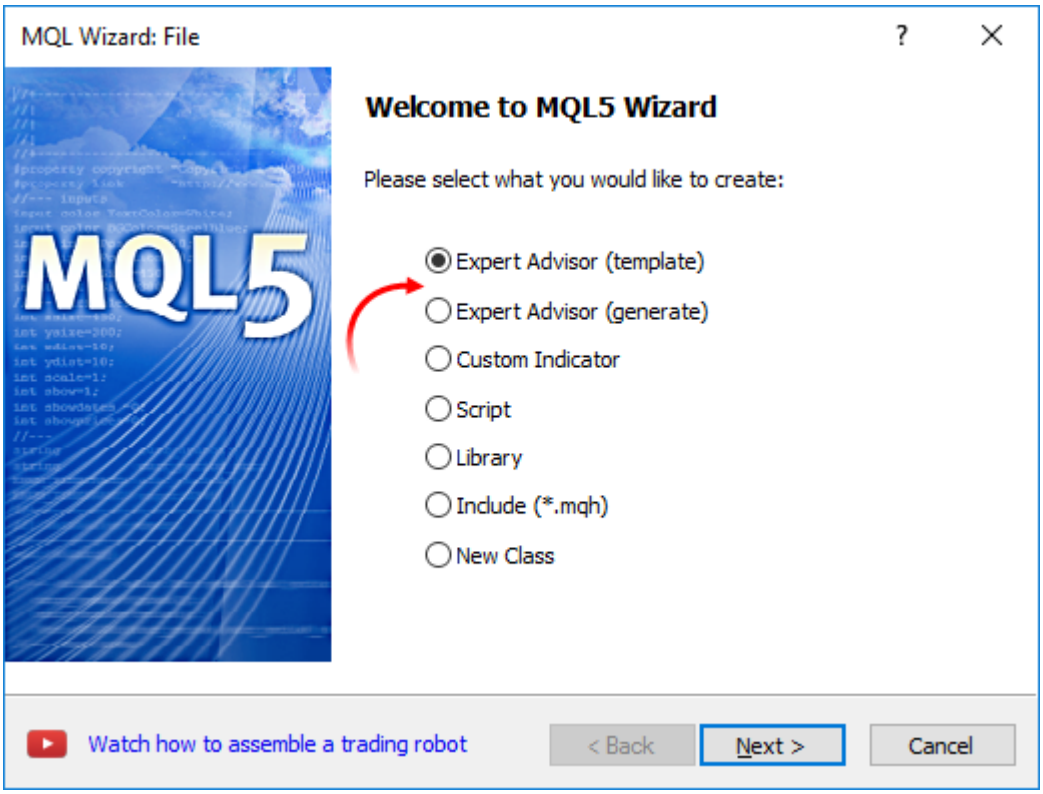
© 2000-2025, MetaQuotes Ltd.

# 交易权限

## 交易自动化

MQL5语言提供了一个专为开发自动交易系统而设计的特别的交易函数组。无人为干涉，为自动交易而开发的程序称为EA交易或自动交易。若要在MetaEditor创建EA交易，请启动MQL5向导并在以下两个选项中选择其中一个：

- Expert Advisor (模板) -允许您用现有的事件处理函数创建一个模板，它应该通过编程，用所有必要的功能进行增补。
- Expert Advisor (生成) -允许您只需选择必要的模块即可开发一个成熟的自动交易：交易信号模块，财务管理模块和追踪止损模块。




交易函数只能在EA交易和脚本中工作。交易不允许用于指标。

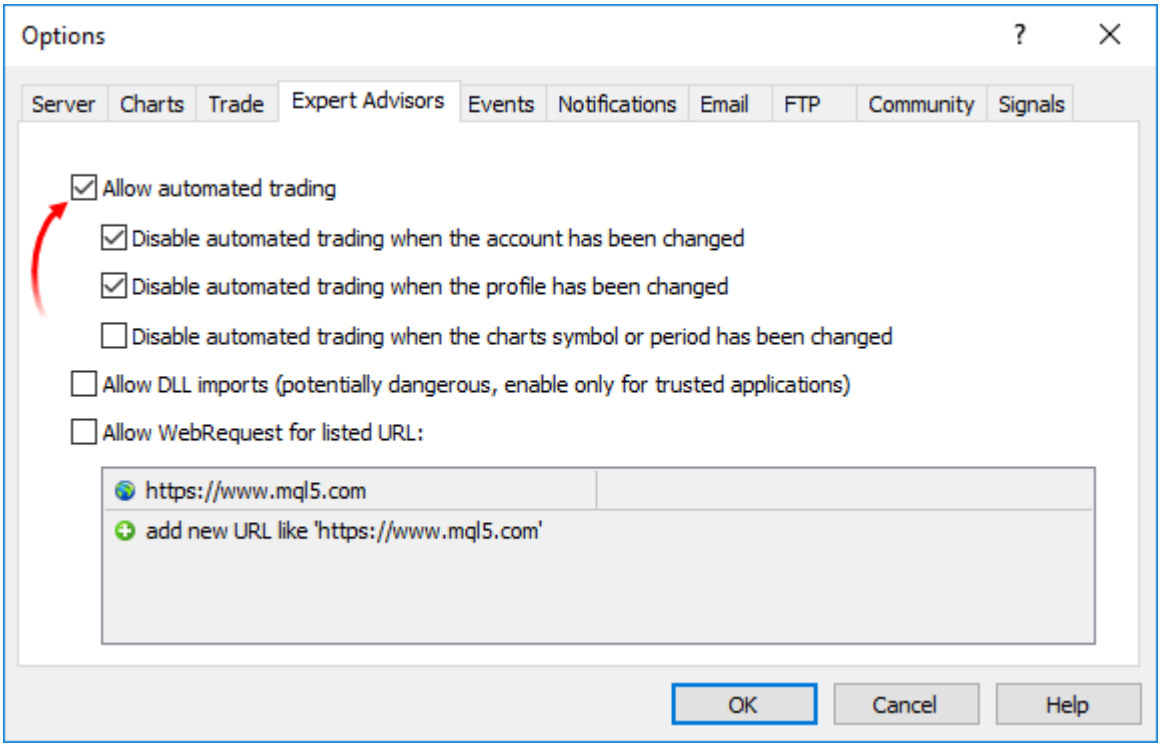
## 检查执行自动交易的权限

若要开发一种可靠的，可以无人为干涉进行工作的EA交易，则需要安排一套重要的检查。首先，如果交易全部允许，我们应该以编程方式进行检查。这是开发任何自动系统时的不可缺少的基本检查。

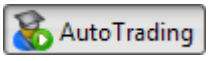
### 检查在程序端执行自动交易的权限

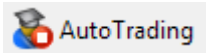
程序端设置为您提供了允许或禁止所有程序自动交易的能力。

© 2000-2025, MetaQuotes Ltd.




您可以在程序端标准面板直接切换自动交易选项：

- 

-启用自动交易，允许使用应用程序中的交易函数。
- 

-禁用自动交易，运行的应用程序不能执行交易函数。

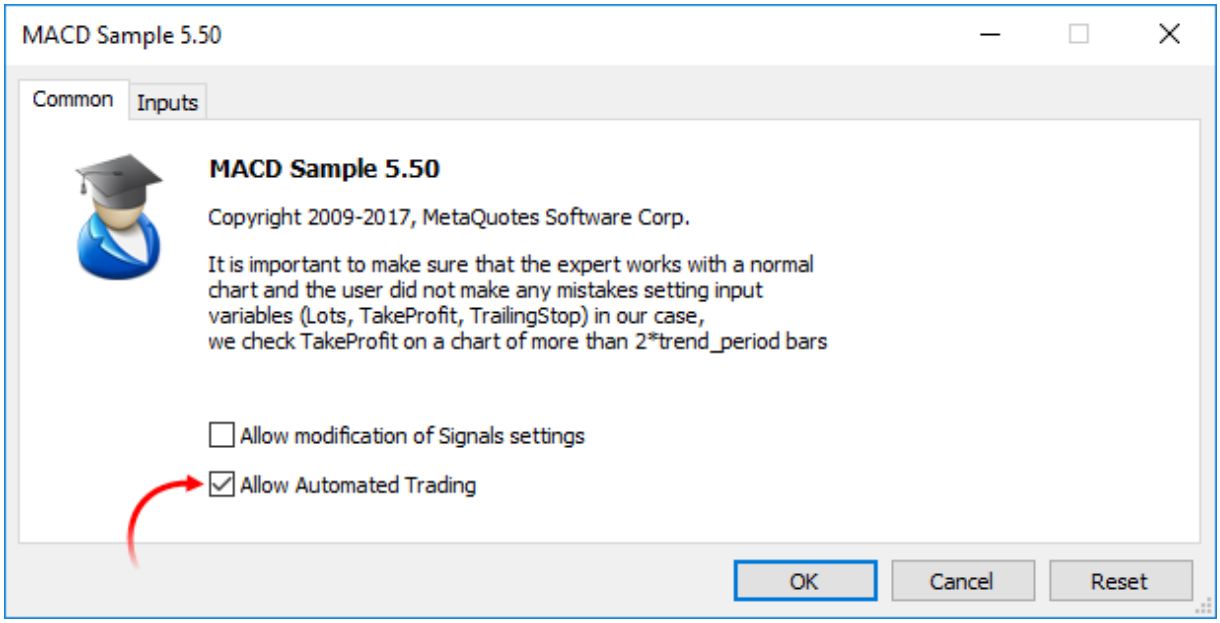
示例检查：

```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
   Alert("Check if automated trading is allowed in the terminal settings!");
```

## 检查是否允许某个运行的EA/脚本进行交易

您可以在启动时允许或禁止某个程序的自动交易。为此，请在程序属性中选择特定的勾选框。

© 2000-2025, MetaQuotes Ltd.




示例检查：

```
if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
   Alert("Check if automated trading is allowed in the terminal settings!");
else
  {
   if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
      Alert("Automated trading is forbidden in the program settings for ",__FILE__)
  }
```

## 检查是否允许当前账户的任何EA/脚本进行交易

在交易服务器端可以禁止自动交易。示例检查：

```
if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT))
   Alert("Automated trading is forbidden for the account ",AccountInfoInteger(ACCOU
   " at the trade server side");
```

如果一个交易账户禁止自动交易，EA交易/脚本的交易操作则不会执行。

## 检查是否允许当前账户进行交易

在某些情况下，某个交易账户禁止任何交易操作 - 既不可以手动执行也不可以自动交易。投资者密码用于连接交易账户时的示例检查：

```
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
   Comment("Trading is forbidden for the account ",AccountInfoInteger(ACCOUNT_LOGIN
         ".\n Perhaps an investor password has been used to connect to the trading
         "\n Check the terminal journal for the following entry:",
         "\n\'",AccountInfoInteger(ACCOUNT_LOGIN),"\': trading has been disabled -
```

AccountInfoInteger(ACCOUNT_TRADE_ALLOWED)在以下情况下可能返回**false**:

- 没有连接交易服务器。可以使用TerminalInfoInteger(TERMINAL_CONNECTED)检查；

© 2000-2025, MetaQuotes Ltd.

- 交易账户切换到只读模式 (发送至存档)；
- 在交易服务器端禁止账户交易；
- 交易账户连接已经在投资者模式下执行。

另见

客户端属性, 账户属性, 运行MQL5程序的属性

© 2000-2025, MetaQuotes Ltd.

# 客户端事件

## Init

在客户端下载程序后（EA交易或者自定义指标）就开始全局变量初始化过程，发送初始化事件，通过OnInit()事件处理器处理过程，当安全性和/或时间表改变时，该程序在MetaEditor编辑器中编译之后，事件生成，EA交易或者自定义指标设置窗口中输入参量改变。在账户改变之后，EA交易也可以初始化，初始化事件不只是为脚本创建的。

## Deinit

在全局变量初始化和程序（EA交易和自定义指标）加载之前，客户端向程序发送Deinit事件。当客户端关闭时，Deinit就会生成，当图表关闭时，安全性和/或时间表改变，成功的程序重编辑，当输入参量改变，账户改变。

无法初始化 可以从参量中获得，通过 OnDeinit() 函数传递，OnDeinit() 函数的运行仅限于2.5秒，如果在此期间，函数没完成，会强制终止，Deinit事件不只为脚本创建的。

## Start

Start是加载脚本或服务后用于启动脚本或服务的特殊事件。它通过OnStart函数处理。Start事件不被传递到EA和自定义指标。

## NewTick

如果有新的引用，NewTick事件会生成，通过EA交易的附加OnTick()过程。当先前引用的OnTick函数正在处理的时候，新的引用就会接收，新的引用会被EA交易忽视，因为类似事件不会列入队伍里。

当程序运行时，接收到的所有新的引用都会忽视，直到OnTick()完成。只有在接收新引用后该函数才会运行，无论自动交易是否允许NewTick事件都会产生（"Allow/prohibit Auto trading"按钮），自动交易引用禁止，EA交易发送的交易请求是不允许的，而EA交易仍旧进行。

自动交易禁令通过按适当按钮会停止当前执行的OnTick()函数，当EA交易属性开始时。

## Calculate

计算事件只在指标完成初始化事件并改变价格数据后开始生成，通过 OnCalculate 函数执行。

## Timer

定时器 事件通过EA交易客户端定期执行，并通过EventSetTimer 函数被定时器激活。通常情况下，该函数通过OnInit调用。定时器事件过程通过OnTimer函数执行。在EA交易操作完成之后，有必要通过EventKillTimer函数删除定时器，通常叫做OnDeinit函数。

## Trade

当交易服务器完成一个交易操作，交易事件生成，交易事件通过 OnTrade() 函数处理，有如下操作步骤：

© 2000-2025, MetaQuotes Ltd.

- 发送、修改或者删除待办订单；
- 因为金额不满或者延期而取消待办订单；
- 激活待办订单；
- 打开，增加或者关闭位置（或者部分位置）；
- 修改打开位置（改变站点-止损数值和/或目标数值）。

## TradeTransaction

当在交易账户上执行一些明确活动时，它的状态会发生变化。这种活动包括：

- 使用 OrderSend 和 OrderSendAsync 函数在客户端从任何MQL5应用发送交易请求及其进一步执行；
- 通过程序端图形界面发送交易请求及其进一步执行；
- 在服务器上激活挂单和止损订单；
- 在交易服务器上执行操作。

这些操作的结果会执行以下交易事务：

- 处理交易请求；
- 改变持仓订单；
- 改变订单历史记录；
- 改变交易历史记录；
- 改变持仓。

例如，当发送市场买入订单时，它会被处理，为账户创建相应的买入订单，然后执行订单并从持仓列表中移除，添加到历史订单，而相应的交易也会添加到历史记录并创建新的持仓。所有这些操作都是交易操作。在程序端到达这种事务就是交易事务事件。该事件通过 OnTradeTransaction 函数处理。

## Tester

测试事件在EA交易测试历史数据完毕后产生，该事件通过 OnTester() 函数处理。

## TesterInit

TesterInit 事件在初次优化通过之前，随着优化的启动，在策略测试中生成。TesterInit事件通过 OnTesterInit()函数处理。

## TesterPass

TesterPass 事件在收到新数据帧时生成。TesterPass事件通过OnTesterPass()函数处理。

## TesterDeinit

TesterDeinit 事件在EA交易优化结束后，在策略测试中生成。TesterDeinit事件通过 OnTesterDeinit()函数处理。

## ChartEvent

© 2000-2025, MetaQuotes Ltd.

当用户使用图表时，客户端产生事件，即ChartEvent：

- 当图表窗口聚集时，按关键字按钮；
- 创建图解对象；
- 删除图解对象；
- 鼠标点击图表的图解对象；
- 使用鼠标移动图解对象；
- 在标签编辑中进行文本末尾编辑；

自定义事件ChartEvent，可以通过使用EventChartCustom函数以任何MQL5程序发送到EA交易中，该事件通过函数 OnChartEvent 处理。

## BookEvent

BookEvent 事件在市场深度改变后通过客户端产生，通过 OnBookEvent函数执行，开始BookEvent程序特定交易对象，有必要通过 MarketBookAdd 函数为该事件订阅交易对象。

为从BookEvent中取消特定的交易品种，调用 MarketBookRelease 函数是有必要的。BookEvent事件是广播类型事件-它表示足以为一个事件订阅一个EA交易，而其他所有EA交易都有OnBookEvent事件，都会接收它。有必要分析交易品种的名称，以一个参量传递处理器。

另见

事件处理程序，程序运行

© 2000-2025, MetaQuotes Ltd.

# 资源

## 在MQL5程序中使用图形和声音

MQL5程序允许使用声音和图形文件：

- PlaySound() 为声音文件；
- ObjectCreate() 允许使用图形对象 OBJ_BITMAP 和 OBJ_BITMAP_LABEL创建用户界面。

### PlaySound()

调用 PlaySound() 函数的示例：

```
//+------------------------------------------------------------------+
//| 调用标准OrderSend() 并播放一段音频                                |
//+------------------------------------------------------------------+
void OrderSendWithAudio(MqlTradeRequest  &request, MqlTradeResult &result)
  {
  //--- 发送一个请求到服务器
   OrderSend(request,result);
   //--- 如果请求被采纳，播放音频Ok.wav
   if(result.retcode==TRADE_RETCODE_PLACED) PlaySound("Ok.wav");
   //--- 如果失败，播放文件timeout.wav的警告音
   else PlaySound("timeout.wav");
  }
```

该示例表达了如何播放来自文件 'Ok.wav' 和 'timeout.wav'的音频，这些文件包含在标准程序包中。这些文件位于 **terminal_directory\Sounds**文件夹。 在这里 **terminal_directory** 是文件夹，MetaTrader 5 客户端在这里启动。程序端目录的位置可以通过以下方式，从mql5程序找到：

```
//--- 文件夹，存储程序端数据的地方
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

您不仅可以使用来自**terminal_directory\Sounds**文件夹的音频文件，还可以使用位于子文件夹**terminal_data_directory\MQL5**的任何音频文件。 您可以通过程序端菜单 “文件”->”打开”程序端数据或使用编程方式找到程序端数据目录的位置：

```
//--- 文件夹，存储程序端数据的地方
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

例如，如果Demo.wav音频文件位于terminal_data_directory\MQL5\Files，那么调用PlaySound() 应该按以下方式编写：

```
//--- 播放来自terminal_directory_data\MQL5\Files\Demo.wav文件夹的Demo.wav
   PlaySound("\\Files\\Demo.wav");
```

请注意在评论中，文件路径使用 “\”反斜杠编写，而在函数中使用 “\\”。

当指定路径时，通常使用双反斜杠作为分隔符，因为单反斜杠是处理程序源代码中的字符串常数和字符常数时，编译器的控制符号。

© 2000-2025, MetaQuotes Ltd.

用NULL参数调用 PlaySound() 函数以停止回放：

```
//--- 用NULL参数调用PlaySound()来停止回放
   PlaySound(NULL);
```

## ObjectCreate()

EA交易的示例，使用ObjectCreate()函数创建一个图形标签 (OBJ_BITMAP_LABEL)。

```
string label_name="currency_label";          // OBJ_BITMAP_LABEL 对象的名称
string euro      ="\\Images\\euro.bmp";      // 文件terminal_data_directory\MQL5\Images\
string dollar    ="\\Images\\dollar.bmp";    // 文件terminal_data_directory\MQL5\Images\
//+------------------------------------------------------------------+
//| EA初始化函数                                                     |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 如果还未创建按钮OBJ_BITMAP_LABEL，请创建
   if(ObjectFind(0,label_name)<0)
     {
      //--- 尝试创建对象OBJ_BITMAP_LABEL
      bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
      if(created)
        {
         //--- 连接图表左上角的按钮
         ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
         //--- 现在设置对象属性
         ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
         ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
         //--- 重置过去错误的代码为0
         ResetLastError();
         //--- 下载图片显示按钮的”按键“状态
         bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
         //--- 测试结果
         if(!set)
           {
            PrintFormat("Failed to download image from file %s. Error code %d",euro,Ge
           }
         ResetLastError();
         //--- 下载图片显示按钮的”无按键“状态
         set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

         if(!set)
           {
            PrintFormat("Failed to download image from file %s. Error code %d",dollar,
           }
         //--- 发送图表评论刷新，以便于无需标记即可立即显示按钮
         ChartRedraw(0);
```

© 2000-2025, MetaQuotes Ltd.

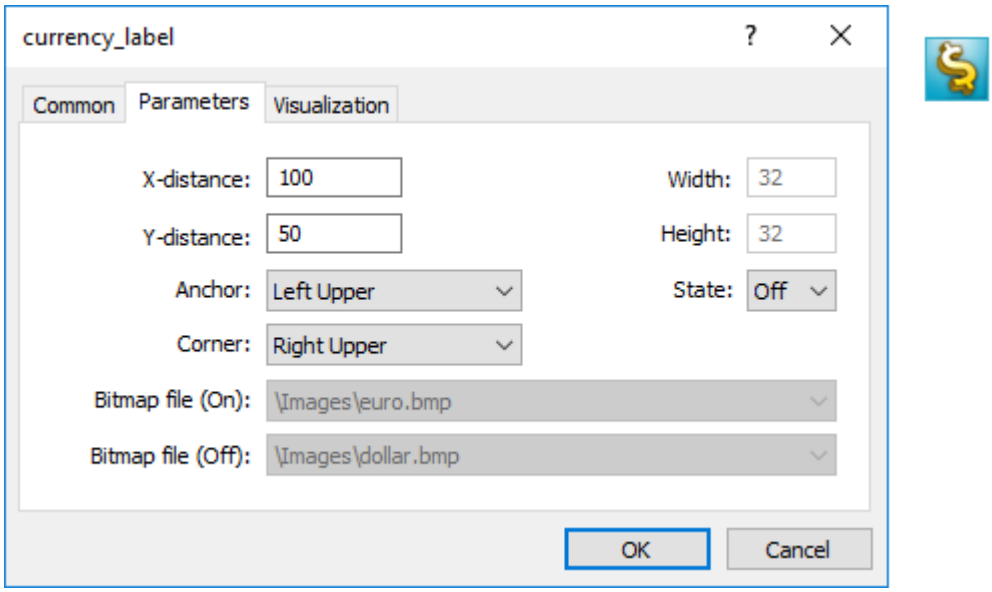
```
         }
       else
         {
          //--- 创建对象失败，公告
          PrintFormat("Failed to create object OBJ_BITMAP_LABEL. Error code %d",GetLast
         }
      }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 删除图表对象
   ObjectDelete(0,label_name);
  }
```

创建和设置名为currency_label的图形对象通过currency_label函数实行。图形文件的路径设置在全局变量 **euro** 和 **dollar**，用双反斜杠作为分隔符：

```
string euro      ="\\Images\\euro.bmp";    // terminal_dara_directory\MQL5\Images\euro
string dollar    ="\\Images\\dollar.bmp";  // terminal_dara_directory\MQL5\Images\doll
```

文件位于文件夹 **terminal_data_directory**\MQL5\Images。

对象 OBJ_BITMAP_LABEL 实际上是一个按钮，根据按钮的状态（已按或未按）展示两个图像中的一个：euro.bmp 或 dollar.bmp。




图形界面的按钮大小会根据图像大小自动调整。通过在OBJ_BITMAP_LABEL对象上点击鼠标左键来改变图像（属性中必须设置"禁用选择"）。OBJ_BITMAP对象以相同方式创建 - 它用于创建必要的背景图像。

© 2000-2025, MetaQuotes Ltd.

OBJPROP_BMPFILE 属性的值，负责对象OBJ_BITMAP 和 OBJ_BITMAP_LABEL的外表，可以动态地变化。这允许为mql5程序创建各种交互式用户界面。

## 包括mql5程序编译期间的可执行文件资源

mql5程序可能需要许多图像和音频文件格式的不同下载资源。为了消除移动MQL5可执行文件时转移所有这些文件的需要，应该会使用到编译器#resource指令：

```
#resource path_to_resource_file
```

#resource 指令告诉编译器指定路径path_to_resource_file的资源应该被包括到可执行EX5 文件。因此，所有这些必要的图像和音频文件可以直接放在EX5文件，以便在您想要在不同程序端运行程序时，无需分别转移其中使用的文件。任何EX5文件都可以包含资源，而任何EX5程序都可以使用来自其他EX5程序的资源。

BMP 和 WAV 格式的文件在包含到EX5文件之前自动压缩。这表示除了创建完整的MQL5程序外，与通常的MQL5编程方式相比较，使用资源还允许在使用图形和音频时，减少所需文件的总大小。

资源文件大小一定不要超过16Mb。

## 通过编译器搜索指定的资源

插入资源使用命令 #resource "<path to the resource file>"

```
#resource "<path_to_resource_file>"
```

常量字符串<path_to_resource_file> 的长度不得超过63个字符。

编译器根据以下顺序在指定路径搜索资源：

- 如果单反斜杠"\"分隔符 (写成"\\")放在路径前面，它搜索目录 terminal_data_directory\MQL5\涉及的资源。
- 如果没有反斜杠，则搜索资源编写所在的源文件的位置涉及的资源。

资源路径不能包括子字符串"..\\"和 ":\\"。

资源包含的示例：

```
//--- 正确的资源规范
#resource "\\Images\\euro.bmp" // euro.bmp 位于 terminal_data_directory\MQL5\Images\
#resource "picture.bmp"        // picture.bmp 位于与源文件相同的目录
#resource "Resource\\map.bmp"  // 资源位于source_file_directory\Resource\map.bmp

//--- 错误的资源规范
#resource ":picture_2.bmp"     // 不能包括 ":"
#resource "..\\picture_3.bmp"  // 不能包括 ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.bmp" //超过
```

© 2000-2025, MetaQuotes Ltd.

# 使用资源

## 资源名称

当资源使用#resource指令声明后，它可以用于程序的任何部分。资源的名称是在行开始没有反斜杠的路径，来设置资源的路径。若要在代码中使用您自己的资源，特殊符号 "::" 应该添加在资源名称的前面。

示例：

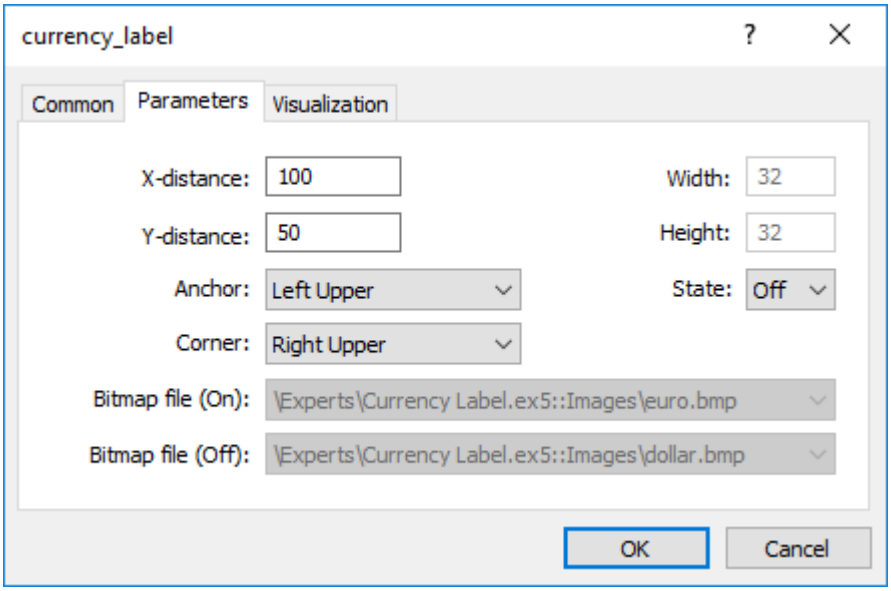
```
//--- 评论中资源规范和其名称的示例
#resource "\\Images\\euro.bmp"           // 资源名称 - Images\euro.bmp
#resource "picture.bmp"                  // 资源名称 - picture.bmp
#resource "Resource\\map.bmp"            // 资源名称 - Resource\map.bmp
#resource "\\Files\\Pictures\\good.bmp" // 资源名称 - Files\Pictures\good.bmp
#resource "\\Files\\Demo.wav";           // 资源名称 - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";        // 资源名称 - Sounds\thrill.wav"
...

//--- 资源利用
ObjectSetString(0,bitmap_name,OBJPROP_BMPFILE,0,"::Images\\euro.bmp");
...
ObjectSetString(0,my_bitmap,OBJPROP_BMPFILE,0,"::picture.bmp");
...
set=ObjectSetString(0,bitmap_label,OBJPROP_BMPFILE,1,"::Files\\Pictures\\good.bmp");
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

请注意，当设置资源到OBJ_BITMAP 和 OBJ_BITMAP_LABEL对象的图像时，OBJPROP_BMPFILE 属性的值不能手动更改。例如，为了创建OBJ_BITMAP_LABEL，我们只有资源 euro.bmp 和 dollar.bmp。

```
#resource "\\Images\\euro.bmp";    // euro.bmp位于terminal_data_directory\MQL5\Images\
#resource "\\Images\\dollar.bmp";  // dollar.bmp位于terminal_data_directory\MQL5\Image
```

当查看该对象属性时，我们将看到属性BitMap 文件 (On) 和 BitMap 文件 (Off) 无效，不能手动更改：

© 2000-2025, MetaQuotes Ltd.




## 使用其他mq5程序的资源

资源使用方面还有另一个优势 - 在任何mq5程序中，也可以使用另一个EX5文件的资源。因此，来自一个EX5文件的资源可以用在许多其他mq5程序。

若要使用来自另一个文件的资源名称，它应该被指定为 <path_EX5_file_name>::<resource_name>。例如，假设Draw_Triangles_Script.mq5 脚本在文件triangle.bmp中包含一个图像资源：

```
#resource "\\Files\\triangle.bmp"
```

然后脚本本身使用的其名称将类似"Files\triangle.bmp"，并且若要使用它，"::" 应该添加到资源名称。

```
//--- 使用脚本资源
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp");
```

若要使用另一个程序的相同资源，例如EA交易，我们需要添加资源名称路径到 **terminal_data_directory\MQL5**相对的EX5文件和脚本EX5文件的名称 - **Draw_Triangles_Script.ex5**。假设脚本位于标准文件夹 **terminal_data_directory\MQL5**Scripts\，那么应该按照以下方式调用：

```
//--- 使用EA中的脚本资源
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triangles_Script.e
```

如果从另一个EX5调用资源时没有指定可执行文件的路径，那么可执行文件在包括调用资源程序的相同文件夹中搜索。这意味着如果EA交易调用来自Draw_Triangles_Script.ex5的资源而没有指定路径，就像这样：

```
//--- 调用未指定路径的EA脚本资源
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.ex5::Files\\
```

那么如果EA位于**terminal_data_directory\MQL5**Experts\，则文件将在文件夹 **terminal_data_directory\MQL5**Experts\中搜索。

© 2000-2025, MetaQuotes Ltd.

## 使用包括资源的自定义指标

一个或多个自定义指标对于MQL5应用程序操作可能是必需的。所有这些都包含在可执行MQL5程序的代码中。包含作为资源的指标简化了应用程序的分布。

下面就是包含和使用位于**terminal_data_folder**\MQL5\Indicators\目录的SampleIndicator.ex5自定义指标的示例：

```
//+------------------------------------------------------------------+
//|                                                     SampleEA.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#resource "\\Indicators\SampleIndicator.ex5"
int handle_ind;
//+------------------------------------------------------------------+
//| EA初始化函数                                                     |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   handle_ind=iCustom(_Symbol,_Period,"::Indicators\\SampleIndicator.ex5");
   if(handle_ind==INVALID_HANDLE)
     {
      Print("Expert: iCustom call: Error code=",GetLastError());
      return(INIT_FAILED);
     }
//--- ...
   return(INIT_SUCCEEDED);
  }
```

OnInit() 函数自定义指标创建一个或多个副本时，这种情况需要特别考虑。请谨记资源应该按照以下方式指明：<path_EX5_file_name>::<resource_name>。

例如，如果SampleIndicator.ex5指标作为资源包含在SampleEA.ex5 EA交易，在自定义指标初始化函数调用iCustom()指定的路径如下： "\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5"。当该路径明确设置时，SampleIndicator.ex5自定义指标严格连接到SampleEA.ex5 EA交易，失去独立工作的能力。

路径本身可以使用GetRelativeProgramPath()函数接收。用法示例如下：

```
//+------------------------------------------------------------------+
//|                                              SampleIndicator.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property indicator_separate_window
#property indicator_plots 0
int handle;
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 提供自身链接的错误方式
//--- 字符串路径="\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5";
//--- 接收自身链接的正确方式
  string path=GetRelativeProgramPath();
//--- 指标缓冲映射
   handle=iCustom(_Symbol,_Period,path,0,0);
   if(handle==INVALID_HANDLE)
     {
      Print("Indicator: iCustom call: Error code=",GetLastError());
      return(INIT_FAILED);
     }
   else Print("Indicator handle=",handle);
//---
   return(INIT_SUCCEEDED);
  }
///....
//+------------------------------------------------------------------+
//| GetRelativeProgramPath                                           |
//+------------------------------------------------------------------+
string GetRelativeProgramPath()
  {
   int pos2;
//--- 接收应用程序的绝对路径
   string path=MQLInfoString(MQL_PROGRAM_PATH);
//--- 找出 "\MQL5\" 字符串的位置
   int    pos =StringFind(path,"\\MQL5\\");
//--- 子字符串未找到 - 错误
   if(pos<0)
      return(NULL);
//--- 跳过 "\MQL5" 目录
   pos+=5;
//--- 跳过多余的 '\' 交易品种
   while(StringGetCharacter(path,pos+1)=='\\')
      pos++;
//--- 如果这是一个资源，返回MQL5目录的相对路径
   if(StringFind(path,"::",pos)>=0)
      return(StringSubstr(path,pos));
//--- 为第一个MQL5子目录找到一个分隔符 (例如，MQL5\Indicators)
//--- 如果没找到，返回MQL5目录相对的路径
   if((pos2=StringFind(path,"\\",pos+1))<0)
      return(StringSubstr(path,pos));
//--- 返回子目录相对的路径(例如， MQL5\Indicators)
   return(StringSubstr(path,pos2+1));
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double& price[])
  {
//--- 返回prev_calculated值用于下次调用
   return(rates_total);
  }
```

## 资源变量

资源可以使用资源变量来声明并被当做适当类型的变量。声明格式：

```
#resource path_to_the_resource_file as resource_variable_type resource_variable_name
```

示例声明：

```
#resource "data.bin" as int ExtData[]              // 声明包含data.bin 文件数据的数字数组
#resource "data.bin" as MqlRates ExtData[]         // 声明包含data.bin文件数据的 示例结构数
//--- 字符串
#resource "data.txt" as string ExtCode             // 声明包含data.txt文件数据的字符串 支持
#resource "data.txt" as string ExtCode[]           // 声明包含data.txt文件字符串的数组 支持
//--- 图形资源
#resource "image.bmp" as bitmap ExtBitmap[]        // 声明包含BMP位图文件的一维数组，数组大小
#resource "image.bmp" as bitmap ExtBitmap2[][]     // 声明包含BMP位图文件的二维数组，数组大小
```

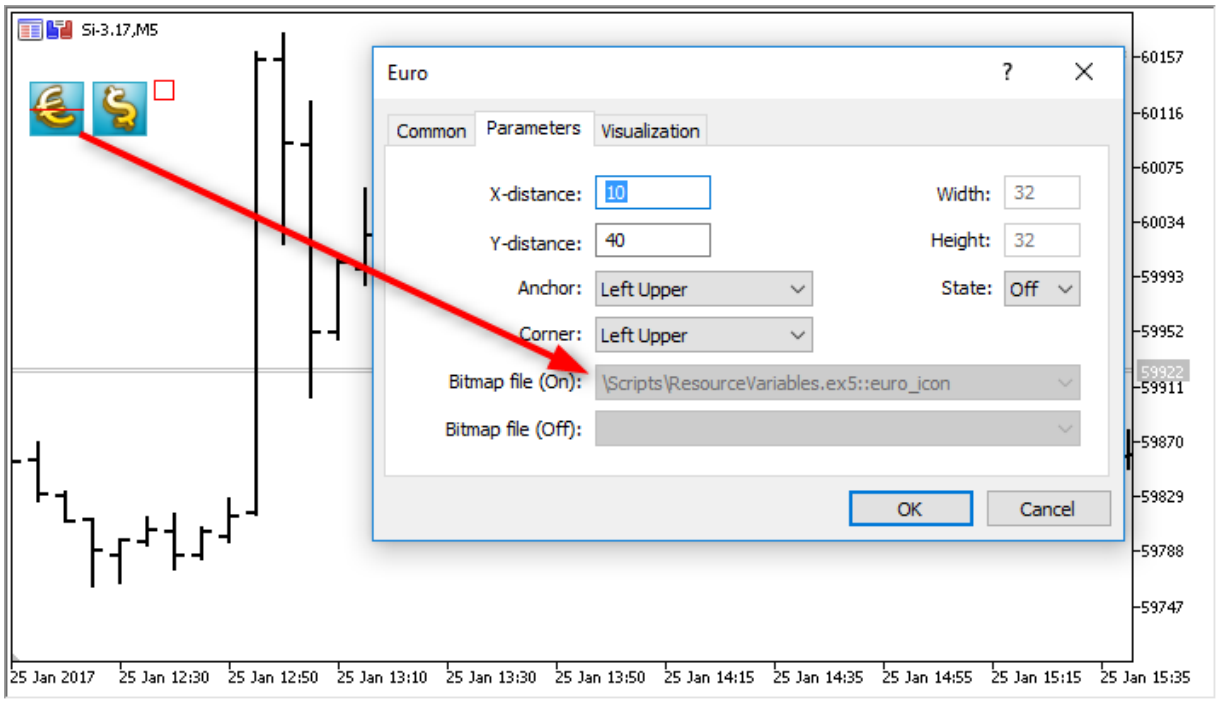
这种声明情况下，资源数据只能通过变量来解决，通过"::<rsource name>"的自动解决 无法工作。

```
#resource "\\Images\\euro.bmp" as bitmap euro[][]
#resource "\\Images\\dollar.bmp"
//+------------------------------------------------------------------+
//|  OBJ_BITMAP_LABEL 对象使用资源创建函数    |
//+------------------------------------------------------------------+
void Image(string name,string rc,int x,int y)
  {
   ObjectCreate(0,name,OBJ_BITMAP_LABEL,0,0,0);
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
   ObjectSetString(0,name,OBJPROP_BMPFILE,rc);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 输出存储在欧元资源变量的图像大小[宽，高]
   Print(ArrayRange(euro,1),", ",ArrayRange(euro,0));
//--- 改变欧元图像 - 在中间绘制红色横纹
   for(int x=0;x<ArrayRange(euro,1);x++)
      euro[ArrayRange(euro,1)/2][x]=0xFFFF0000;
//--- 使用资源变量创建图形资源
   ResourceCreate("euro_icon",euro,ArrayRange(euro,1),ArrayRange(euro,0),0,0,ArrayRang
//--- 创建欧元图形标签对象，并在此设置euro_icon资源图像
   Image("Euro","::euro_icon",10,40);
//--- 应用资源的另一个方法，我们不能绘制
   Image("USD","::Images\\dollar.bmp",15+ArrayRange(euro,1),40);
//--- 不提供解决euro.bmp资源的直接方法，因为它已经通过欧元资源变量进行声明
   Image("E2","::Images\\euro.bmp",20+ArrayRange(euro,1)*2,40); // 将会发生执行时间错误
  }
```

脚本执行结果 -三个对象中只能创建两个OBJ_BITMAP_LABEL对象。第一个对象的图像在中间有红色条纹。




应用资源的重要优势就是资源文件在编译成EX5可执行文件之前自动解压。因此，使用资源变量可以使您将全部必要数据直接放入可执行文件EX5以及减少相较于MQL5程序传统编写方式的文件的数量和总大小。

使用资源变量可以非常方便在市场发布产品。

## 特点

- 特殊的*位图*资源变量类型告诉编译者资源使一个图像。这种变量接收uint类型。
- *位图*类型数组资源变量可以有两种维度。在这种情况下，数组大小定义为[image_height ][image_width]。如果指定了一维数组，那么元素数量则等于image_height*image_width。
- 下载24-位图像时，阿尔法通道组件的全部图像像素设置为255.

© 2000-2025, MetaQuotes Ltd.

- 下载没有阿尔法通道的32-位图时，阿尔法通道组件的全部图像像素设置为255。
- 下载有阿尔法通道的32-位图时，不以任何方式处理像素。
- 资源文件大小不能超过128 Mb。
- 通过存在的BOM（标题）自动编码检测执行字符串文件。如果没有BOM，则通过文件内容定义编码。支持ANSI，UTF-8和UTF-16编码文件。当阅读文件数据时，全部字符串转换为Unicode。

## OpenCL程序

使用资源字符串变量可以极大的促进一些程序的开发。例如，您可以在独立的CL文件内编写一个OpenCL 程序的编码，然后将其包含为一个字符串放入您的MQL5程序资源。

```
#resource "seascape.cl" as string cl_program
...
int context;
if((cl_program=CLProgramCreate(context,cl_program)!=INVALID_HANDLE)
  {
   //--- 通过OpenCL程序执行进一步操作
  }
```

在这个示例中，如果没有使用*cl_program*资源变量，那么您需要编写整个代码作为一个大的字符串。

另见

ResourceCreate(), ResourceSave(), PlaySound(), ObjectSetInteger(), ChartApplyTemplate(), 文件函数

© 2000-2025, MetaQuotes Ltd.

# 调用输入函数

在执行MQL5程序的过程中，客户端较早使用输入函数订阅。这意味着，如果服务器调用输入函数，在程序加载过程中，类似模块(ex5 or dll)就会装载，MQL5和DLL数据库通过调用模块执行。

不推荐使用模块的全部定义名称来加载，比如*Drive:\Directory\FileName.Ext*。MQL5数据库从*terminal_dir\MQL5\Libraries*文件夹中加载，如果数据库没有找到，客户端会尝试加载到*terminal_dir\experts*文件夹中。

系统数据库（DLL）通过操作系统规则加载，如果数据库已经加载完毕（例如，另外的EA交易，甚至是其他的客户端，并联在其中运行），然后到已经建立好的数据库中使用要求，否则，它会按如下顺序搜索：

1. 目录，从模块输入点dll开始,该模块是EA交易，脚本，指标或者EX5数据库；
2. 目录 terminal_data_directory\MQL5\Libraries （TERMINAL_DATA_PATH\MQL5\Libraries）；
3. 目录，MT5客户端开始启动；
4. 系统目录；
5. 窗口目录；
6. 当前目录；
7. PATH系统变量的目录列表。

如果DLL数据库使用另外的DLL工作，当没有第二个DLL时，第一个不能下载。

在EA交易（脚本，指标）下载后，所有EX5数据库模块的普通列表建立。通过使用加载的EA交易（脚本，指标）和列表数据库，之前加载过多次使用的EX5数据库模块是需要的，数据库使用EA交易（脚本，指标）中的 预定义变量 来调用。

输入数据库EX5在如下列表中搜寻：

1. 目录，与建立的EA交易（脚本，指标）路径相关，输入EX5；
2. 目录terminal_directory\MQL5\Libraries；
3. 目录MQL5\Libraries在所有MT5客户端的共同目录中（Common\MQL5\Libraries）。

函数 imported DLL在MQL5程序中一定要确定API窗口允许使用，为了确保同意，文本程序添加到C或者C++，使用关键字__stdcall，针对 Microsoft(r) 编译器，该协议具有如下特性：

- 访客（在我们系统中是MQL5程序）应该注意到函数调用的原型（从DLL中输入），为了适当结合参量堆积；
- 访客（在我们系统中是MQL5程序）以倒序方式将参量进行堆积，从右至左-在此命令中，输入函数从参量中传递；
- 参量通过值传递，除了通过引用明确传递（在我们这一排）；
- 通过阅读参量传递，输入函数独立清除堆积。

当描述输入函数的原型，可以使用默认参量。

如果类似数据库不能加载，或者DLL禁止使用，或者输入函数没找到-EA交易在分目录（子文件）发出消息“EA交易停止”来停止操作。EA交易在重启后会自动运行，在属性表格打开或者按下确认键之后，EA交易重启后重新编译。

# 传递参量

© 2000-2025, MetaQuotes Ltd.

所有 简单型 参量通过值传递，除非明确表明通过引用传递，当字符串传递时，缓冲区的地址复制字符串传递，如果字符串通过引用传递，字符串缓冲区的地址不用复制就能从DLL传递输入函数。

结构包括动态数组，字符串，分类和其他复杂结构，和列举对象的静态或者动态数组，不能通过输入函数传递参量。

当传递数组到DLL，数据缓冲区开头的地址总是能传递（不考虑AS_SERIES标签）。在DLL中的函数与AS_SERIES标签无关，传递数组是不明确长度的静态数组，规定数组大小应使用额外参量。

© 2000-2025, MetaQuotes Ltd.

# 运行时间出错

客户端执行子系统在MQL5程序运行中任何情况下都有机会存储错误代码，对于每个可执行的MQL5程序来说都有预定义变量_LastError。

在开始 OnInit 函数时，重置_LastError变量，在计算或者内部函数的调用过程中，任何错误情况都会发生，_LastError变量接收类似错误代码，存储在变量中的值可以通过 GetLastError() 函数获得。

有几个关键性误差，迅速出现在程序终端：

- 除以0；
- 超出数组范围；
- 使用错误 对象指针 ；

© 2000-2025, MetaQuotes Ltd.

# 测试交易策略

自动交易的理念是每周7天，全天候24小时的无间断自动交易。自动交易不会疲惫，受质疑或让人恐惧，它完全脱离了任何心理问题。只要清楚形式化交易规则并在算法中实施就已经足够，并且自动交易可以持续性无疲惫感的工作。但是，首先，您必须确保满足以下重要的两点：

- EA交易根据交易系统的规则执行 交易操作；
- EA实施的交易策略，在历史记录上显示利润。

若要获得这些问题，我转到MetaTrader5客户端的 策略测试 。

这部分涵盖程序测试的特点和策略测试的优化：

- 订单号生成模式
- 模拟点差
- 测试期间使用真实报价
- 客户端全局变量
- 测试期间的指标计算
- 测试期间加载历史记录
- 多币种测试
- 策略测试的模拟时间
- 测试的图形对象
- 策略测试的 OnTimer() 函数
- 策略测试的Sleep() 函数
- 策略测试的 Print() 函数
- 使用策略测试用于数学运算中的优化问题
- 在 "只开盘价"模式同步柱形
- 测试器的IndicatorRelease() 函数
- 测试器中的事件处理
- 测试代理
- 程序端和代理之间的数据交换
- 使用所有客户端的共享文件夹
- 使用 DLLs

## MQL5云网络中的内存和磁盘空间限制

以下限制适用于在MQL5云网络中运行的优化：EA交易不得向磁盘写入超过1GB的信息或使用超过4GB的内存(RAM)　　。如果超出限制，网络代理将无法正确完成计算，您将不会收到结果。但是，您需要为计算所花费的所有时间付费。

如果您需要从每个优化过程获取信息，请发送帧而不写入磁盘。为了避免在MQL5云网络计算期间在EA交易中使用文件操作，您可以使用以下检查：

```
int handle=INVALID_HANDLE;
```

© 2000-2025, MetaQuotes Ltd.

```
bool file_operations_allowed=true;
if(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_FORWARD))
   file_operations_allowed=false;

if(file_operations_allowed)
  {
   ...
   handle=FileOpen(...);
   ...
  }
```

## 策略测试中的函数限制

客户端的策略测试中，对一些函数有操作限制。

### Comment()，Print() 和 PrintFormat() 函数

为了提高性能，优化自动交易参数时不能执行 Comment()，Print() 和 PrintFormat() 函数。除非在OnInit()处理程序内部使用这些函数。这可以使您在错误发生时，轻松找到错误的原因。

### Alert()，MessageBox()，PlaySound()，SendFTP，SendMail()，SendNotification()，WebRequest() 函数

专为与“外部世界”互动而设计的 Alert()，MessageBox()，PlaySound()，SendFTP()，SendMail()，SendNotification() 和 WebRequest() 函数不在策略测试中执行。

## 订单号生成模式

EA交易是一个用MQL5编写的程序，每次运行都会反映一些外部事件 。EA有一个对应的函数(event handler)，用于每次预定义事件。

NewTick 事件（价格变化）是EA的主要事件，因此，我们需要生成一个订单号序列来测试EA。MetaTrader5客户端的策略测试中共实施三种订单号生成模式：

- 每个订单号
- 1 分钟 OHLC（分钟柱的OHLC 价）
- 只有开盘价

最基本最详细的就是“每个订单号”模式，其他两种模式是该基本模式的简化，并且将被对应“每个订单号”模式进行描述。考虑这三种模式，以便于了解其中的不同。

### "Every Tick"

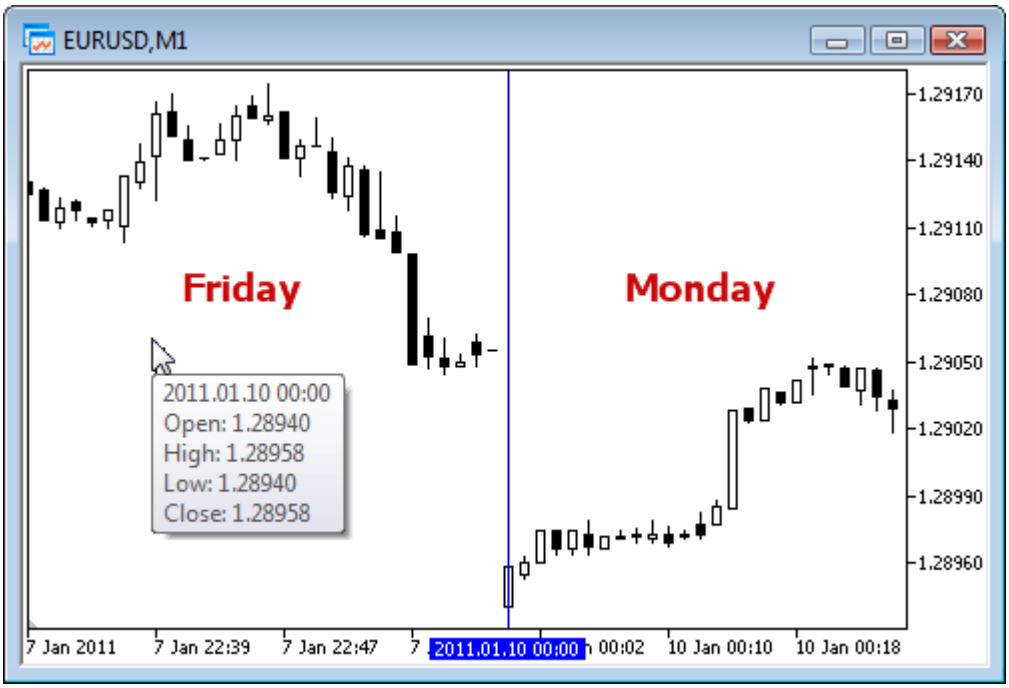
金融工具的历史报价数据以打包分钟柱的形式从交易服务器转移到MetaTrader 5 客户端。发生请求和建设所需时间帧的详细信息可以从MQL5参考的组织数据访问 章节获得。

© 2000-2025, MetaQuotes Ltd.

价格历史的最小元素就是分钟柱，从这里您可以获得四个价格值的信息：

- Open - 分钟柱的开盘价；
- High - 该分钟柱期间的最高价；
- Low - 该分钟柱期间的最低价；
- Close - 分钟柱的收盘价。

新的分钟开始时不会打开新的分钟柱（秒数归零），但是订单号产生时 - 价格变化至少一点时。数字显示新交易周的首个分钟柱，开盘时间是2011.01.10 00:00。我们在图表上看到的周五和周一的价格差，很普遍，因为汇率甚至周末的波动以反馈收到的新闻。



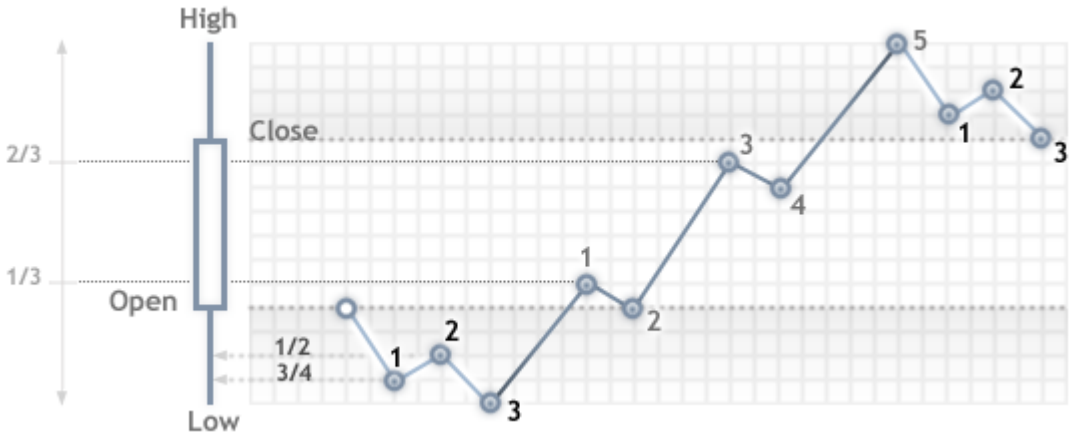

对于该柱，我们只知道分钟柱在2011.01.10的零点零分打开，但是我们不知道秒数。它可以在00:00:12或00:00:36打开（新一天后的12或36秒）或在那分钟的任何其他时间。但是我们知道EURUSD在新分钟柱开盘时间的开盘价是1.28940 。

我们也不知道一秒钟内，何时收到受考虑分钟柱的收盘价的订单号。我们只知道一件事 - 分钟柱的最近收盘价。该分钟内，价格是1.28958。最高价和最低价出现的时间也不知道，但是我们知道最高价格和最低价格分别为 1.28958 和 1.28940水平。

要测试交易策略，我们需要一系列订单号，将在这里模拟EA交易。因此，对于每个分钟柱，我们都知道 4 个控制点，在这里价格可以清楚显示。如果一个柱只有4个订单号，那么就足够执行一个测试，但是通常订单交易量要大于4。

因此，在开盘价，最高价，最低价和收盘价之间，需要生成额外的订单号控制点。"每个订单号"的订单号生成模式的原则显示在 MetaTrader 5 程序端策略测试器中的订单号生成算法 ，数字显示如下。

© 2000-2025, MetaQuotes Ltd.




在“每个订单号”模式中测试时，EA的 OnTick() 函数在每个控制点都将会调用。每个控制点都是生成序列的订单号。EA将会接收模拟订单号的时间和价格，就如在线工作一样。

**重要：** “每个订单号”测试模式是最精确的，但是同时也是最耗时的。对于主要交易策略的初始测试，使用其他两个测试模式中的一个就已经够用。

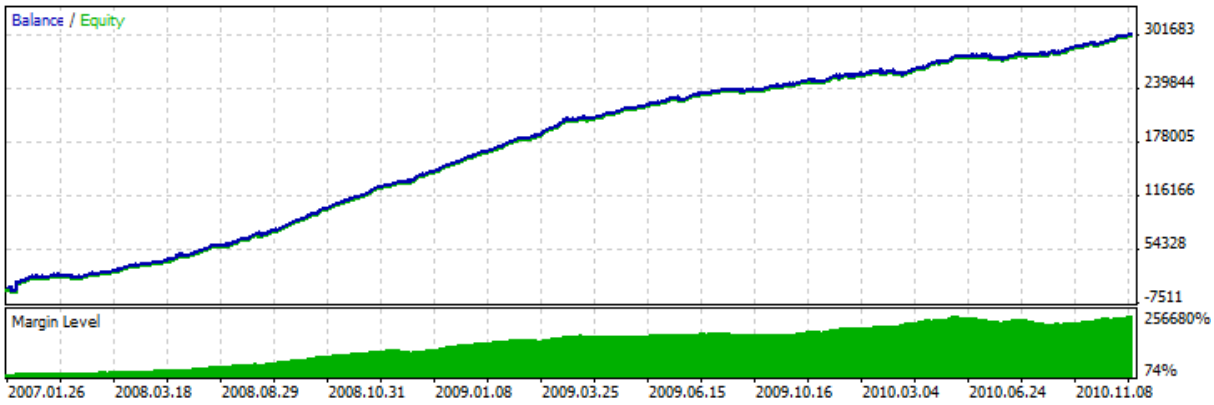
## "1 分钟 OHLC"

“每个订单号”模式是三个模式中最精确的，但是同时，也是最慢的。运行 OnTick() 处理程序在每个订单号都会产生，订单交易量可以非常巨大。对于一个策略，订单号价格波动序列并不重要，因为有更快更粗糙的模拟模式 - "1分钟 OHLC"。

在 "1 分钟 OHLC" 模式，订单号序列仅通过**分钟柱的OHLC价格**搭建，生成的控制点数量也会大量减少 - 因此，这是测试时间。在所有控制点上启动OnTick () 函数，这个可以通过OHLC分钟柱的价格来搭建。

拒绝在开盘价，最高价，最低价和收盘价之间生成额外的中间订单号，会导致出现从开盘价决定时刻起的价格发展的刚性决定论。这使得可以创建“测试盘”，它显示了一个不错的向上的测试平衡图形。

该大盘在代码库中显示的示例 - Grr-al.




数字显示的EA测试图形极具吸引力。它是如何获得的呢？我们知道一个分钟柱有4个价格，并且我们也知道第一个是开盘价，最后的是收盘价。两个价格中间，我们有最高价和最低价，而

© 2000-2025, MetaQuotes Ltd.

发生的顺序我们不得而知，但是我们知道，最高价大于或等于开盘价（而最低价小于或等于开盘价）。

它已足以确定接收开盘价的时间，然后分析下一个订单号以便决定我们此刻的价格 - 最高价还是最低价。如果价格低于开盘价，那么这是最低价并在此刻买入，下一个订单号将相当于最高价，在这里我们关闭购买打开卖出。下一个订单号是最后一个，就是收盘价，我们关闭上面的卖出交易。

如果此价格后，我们收到价格大于开盘价的订单号，那么交易顺序相反。在该"欺骗"模式处理分钟柱，并等候下一个。

当在历史记录测试这种EA时，一切进展都很顺利，但是一旦在线启用，真实情况就开始显露 - 平衡线保持稳定，但向下走。若要识破这个诡计，我们只需在"每个订单号"模式运行EA。

**注意：** 如果粗略测试模式下的EA测试结果（"1分钟OHLC"和"只有开盘价"）似乎很好，一定要在"每个订单号"模式再测试一次。

## "只有开盘价"

在该模式下，订单号的生成基于选为测试的时间帧的OHLC价格。EA交易的OnTick() 函数只在柱开始的开盘价运行。鉴于这个特点，止损水平和挂单可能会在不同于指定价格的价格点触动（尤其在高时间帧测试时）。相反，我们有机会快速运行EA评估测试。

W1和MN1周期在"只有开盘价"订单号生成模式是个例外：这些时间帧的订单号在每日的OHLC价格生成，而非每周或每月的OHLC价格。

假定我们在"只有开盘价"模式测试EURUSD H1的EA交易。在这个情况下，测试间隔内，订单号总数（控制点）将不会超过4*小时柱数量。但是**OnTick() 处理程序调用仅在打开小时柱的时候**。正确测试所需的检查在其他订单号时期出现（从EA"隐藏"）。

- 计算所需预付款；
- 启动止损和获利水平；
- 启动挂单；
- 移除过期的挂单。

如果没有持仓或挂单，我们无需在隐藏订单号执行这种检查，提高速度可能就是安静的本质。该"只有开盘价"模式很好的适应了测试策略，处理交易仅在柱的开始进行，而不使用挂单，以及止损和获利订单。对于这种策略的类，保护必要的测试精确性。

让我们使用来自标准组件的移动平均EA交易作为EA示例，它可以在任何模式下被测试。该EA按照所有决策都在开柱时制定的方式建立，并且立即执行交易，无需使用挂单。

在2010.09.01到2010.12.31期间运行EURUSD H1的EA测试，并比较图形。数字显示了所有三种模式测试报告的结余图。

© 2000-2025, MetaQuotes Ltd.

如您所见，不同测试模式的图形与标准组件的移动平均EA的图形完全相同。

在 "只有开盘价"模式有一些限制：

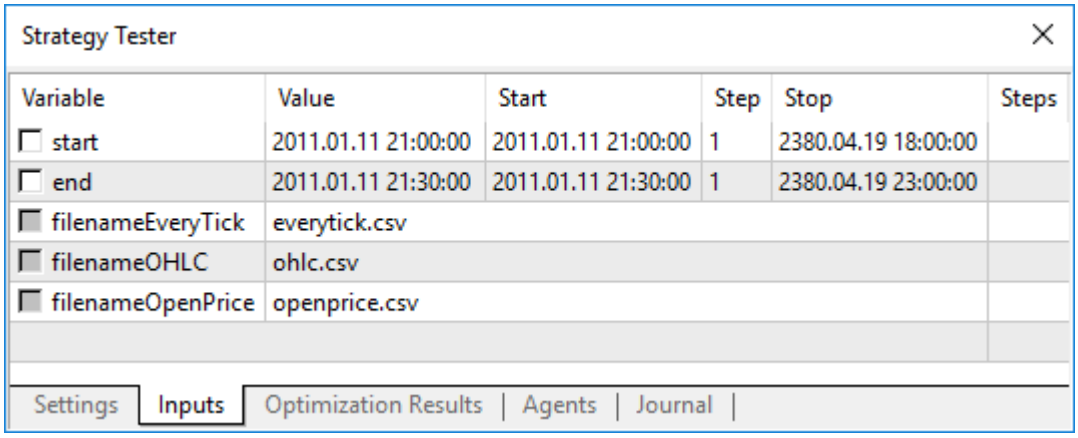
- 您不能使用 随机延迟执行模式。
- 在测试的EA交易，您不能访问低于用于测试/优化的时间帧数据。例如，如果您在H1周期运行测试/优化，您可以访问H2，H3，H4等等的数据，但不可以访问M30，M20，M10数据。另外，访问的高级时间帧也必须是测试时间帧的倍数。例如，如果您在M20运行测试，您不能访问M30的数据，但是可以访问H1。这些限制与不能获得较低的数据或测试/优化期间生成的非倍数时间帧有关。
- 访问其他时间帧数据的限制也用于其他EA事宜的交易品种数据。在这个方面，每个交易品种的限制取决于测试/优化期间访问的第一个时间帧。假如，EURUSD H1测试期间，EA交易访问GBPUSD M20数据。这种情况下，EA交易将能够进一步使用EURUSD H1，H2，以及GBPUSD M20，H1，H2等等。

> **注意：** "只有开盘价"模式是最快的测试方式，但是不适合所有的策略测试。在交易系统的特点上选择所需的测试模式。

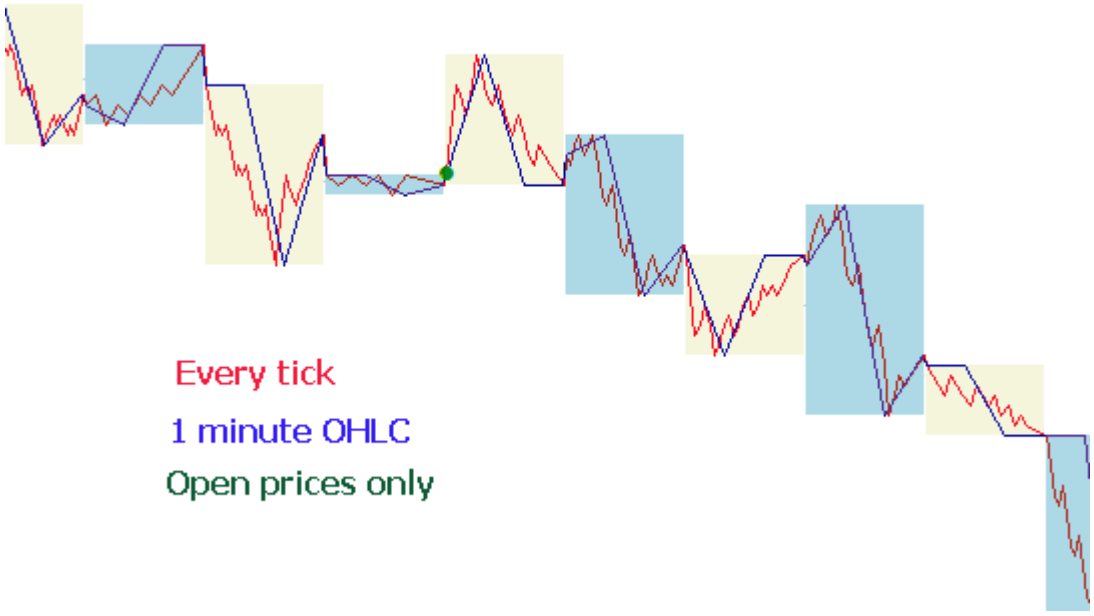
要总结订单号生成模式，请考虑EURUSD不同订单号生成模式的视觉比较，2011.01.11 21:00:00 - 2011.01.11 21:30:00之间的两个M15柱。

使用WriteTicksFromTesterm q5 EA将订单号保存在不同的文件里并且这些文件名称的结尾指定了filenameEveryTick，filenameOHLC 和 filenameOpenPrice 导入参数。

© 2000-2025, MetaQuotes Ltd.




若要使用三个订单号序列（每个以下模式"每个订单号"，"1分钟OHLC"和"只有开盘价"）获得三个文件，EA在相应模式中启动三次。那么，来自这三个文件的数据展示在使用TicksFrom Testerm q5指标的图表上。指标代码附属于该文章。




默认情况下，MQL5语言的所有 文件操作 都在"文件箱"中完成，测试EA过程中只访问它自己的"文件箱"。为了使指标和EA测试期间使用来自一个文件夹的文件，我们使用 flag FILE_COMMON。EA的代码示例：

```
//--- 打开文件
   file=FileOpen(filename,FILE_WRITE|FILE_CSV|FILE_COMMON,";");
//--- 检查文件句柄
   if(file==INVALID_HANDLE)
     {
      PrintFormat("Error in opening of file %s for writing. Error code=%d",filename,Ge
      return;
     }
   else
     {
      PrintFormat("The file will be created in %s folder",TerminalInfoString(TERMINAL_
```

© 2000-2025, MetaQuotes Ltd.

```
   }
```

若要阅读指标中的数据，我们也使用flag FILE_COMMON。这使得我们可以避免从一个文件夹手动转移必要的文件到另一个文件夹。

```
//--- 打开文件
   int file=FileOpen(fname,FILE_READ|FILE_CSV|FILE_COMMON,";");
//--- 检查文件句柄
   if(file==INVALID_HANDLE)
     {
      PrintFormat("Error in open of file %s for reading. Error code=%d",fname,GetLastE
      return;
     }
   else
     {
      PrintFormat("File will be opened from %s",TerminalInfoString(TERMINAL_COMMONDATA
     }
```

## 模拟点差

卖价买价之间的不同价格称为点差。测试期间，点差不能模仿但可以从历史数据获得。如果历史数据中点差少于或等于零，那么最后知道的点差（生成的时刻）被测试代理使用。

在策略测试中，点差通常被认为都是浮动的。所以SymbolInfoInteger(交易品种，SYMBOL_SPREAD_FLOAT)总是返回true。

另外，历史数据包含跳动量的值和交易量。若要存储和检索数据，我们使用特殊的 MqlRates 结构：

```
struct MqlRates
  {
   datetime time;         // 周期开始时间
   double   open;         // 开盘价
   double   high;         // 周期的最高价
   double   low;          // 周期的最低价
   double   close;        // 收盘价
   long     tick_volume;  // 跳动量
   int      spread;       // 点差
   long     real_volume;  // 交易量
  };
```

## 测试期间使用真实报价

使用真实报价进行测试和优化能够尽可能的贴近真实条件。除了基于分钟数据生成报价以外，还可以使用交易商累积的真实报价。也就是来自交易所和流通量提供商的报价。

为了确保测试的最大精确性，分钟柱也用于真实报价模式。这些柱可以检查和纠正报价数据。也可以使您避免测试器和客户端图表之间的差异。

© 2000-2025, MetaQuotes Ltd.

测试器会对比报价数据和分钟柱参数：报价不应超过柱的最高价/最低价，同时初始价和最终价应该符合柱的开盘价/收盘价。交易量也会进行比对。如果检测到任何不匹配，该分钟柱的所有报价都将被放弃。用生成的报价来替代（类似"每个报价"模式）。

如果交易品种历史记录是没有报价数据的分钟柱，测试器会以"每个报价"模式生成报价。这可以在交易商报价数据不足的情况下，在测试器中绘制正确的图表。

如果交易品种历史记录没有分钟柱但存在分钟报价数据，那么该数据会被用于测试器。例如，使用最后价形成交易所的交易品种组。如果服务器只收到卖价/买价而没有最后价，则不会生成柱形图。测试器会使用这些报价数据，因为它们与分钟柱并不抵触。

报价数据可能因为种种原因不符合分钟柱，例如由于连接损耗或从源位置到客户端迁移数据失败。分钟柱的数据在测试中被认为更加可靠。

真实报价测试时请牢记以下特性：

- 启动测试时，交易品种的分钟数据与报价数据同步。
- 报价存储在策略测试器的交易品种缓存中。缓存大小不可超过128 000个报价。收到新报价时，旧数据会从缓存删除。但是，CopyTicks函数允许接收缓存外的报价（仅适用于真实报价测试的情况下）。这样，从测试器的报价数据库请求的数据完全类似于对应的客户端数据库。该数据库不执行分钟柱更正。因此，这里的报价可能不同于缓存中存储的报价

## 客户端全局变量

测试期间，也会模仿客户端全局变量，但是它们与当前的程序端全局变量无关，可以在程序端使用F3按钮查看这种情况。它意味着程序端全局变量的所有操作，在测试期间，都可以超出客户端进行（测试代理）。

## 测试期间的指标计算

在实时模式下，在每次报价计算指标值。

在策略测试中，指标仅在访问数据时计算，即在请求指标缓冲区值时。唯一例外就是具有指定#property tester_everytick_calculate的自定义指标。在这种情况下，每次报价都会重新计算。

在可视化测试模式下，当新报价到达时，所有指标都会无条件地重新计算，以便正确显示在可视化测试图表上。

指标在每次报价计算一次。在新报价到达之前，所有后续对指标数据的请求都不会导致重新计算。因此，如果在EA中通过EventSetTimer()函数启用计时器，则从每次调用OnTimer()处理程序之前最后一次报价请求指标数据。如果尚未在最后一次报价计算指标，则启动指标值的计算。如果数据已经准备好，则无需重新计算即可提供。

因此，所有指标计算都以最节省资源的方式执行 — 如果指标已经以给定报价计算，则其数据按"原样"提供。不会启动重新计算。

## 测试期间加载历史记录

将要测试的交易品种的历史记录在开始测试进程之前会通过交易服务器的程序端进行同步和加载。在第一次的时候，程序端会加载所有可用的交易品种历史记录，以避免随后的请求。之后只需加载新数据即可。

© 2000-2025, MetaQuotes Ltd.

测试开始以后，测试代理直接从客户端接收要被测试的交易品种历史记录。如果在测试过程中使用了其他工具的数据（例如，多货币的EA交易），测试代理则在第一次调用该数据时从客户端请求所需的历史记录。如果程序端提供历史数据，它们会立刻传到测试代理。如果不提供数据，程序端会从服务器请求和下载数据，然后传到测试代理。

其他工具的数据也被需要用于计算交易操作的交叉汇率。例如，用存款货币USD测试EURCHF策略时，处理第一笔交易操作之前，测试代理要求来自客户端的EURUSD和USDCHF历史数据，虽然策略不包含直接使用调用这些交易品种。

测试多货币策略之前，建议下载所有必要的历史数据到客户端。这将有助于避免与下载这些所需数据有关的测试/优化延迟。您可以下载历史记录，例如，通过打开相应的图表并滚动到最开始的历史记录。MQL5参考的组织访问数据部分会提供强制加载历史记录到程序端的示例。

测试代理，反过来，会通过打包的形式接收来自程序端的历史记录。下一次测试期间，测试器就不会加载程序端的历史记录，因为所需数据已经从之前运行测试器的时候获得。

- 程序端仅从交易服务器加载一次历史数据，首次时候，代理从程序端请求要被测试的交易品种的历史记录。历史记录以打包的形式加载以减少流量。
- 订单号不会发送至网络，它们会在测试代理生成。

## 多货币测试

策略测试器允许我们执行多交易品种交易策略的测试。这种EA按惯例被称为多货币EA交易，因为最初，在早先的平台中，只能为一个单一的交易品种执行测试。而在MetaTrader 5程序端的策略测试器中，我们可以模拟所有可用交易品种的交易。

第一次调用交易品种数据时，测试器会自动加载来自**客户端**（而非交易服务器！）的所用交易品种的历史记录。

测试代理仅下载丢失的历史记录，以小预付款提供历史记录所需的数据，用于在测试开始时计算指标。对于时间帧D1和更少，下载历史最少量为一年。

因此，如果我们在2010.11.01-2010.12.01期间以M15（每柱等于15分钟）为周期运行测试（测试间隔一个月），那么程序端将会请求工具2010一整年的历史记录。对于周时间帧，我们将要求100柱的历史记录，这将是两年的时间（一年52周）。对于月时间帧的测试，代理将请求8年的历史记录（12月x8年=96月）。

如果不是必要的柱，测试之前，**测试开始日期将会自动从过去转换到现在**提供必要的柱储备。

测试期间，也模仿"市场报价"，从哪一个能获得交易品种信息.

默认下，测试开始时，策略测试器的"市场报价"中只有一个交易品种 - 测试运行的交易品种。被提及时，所有必要的交易品种都自动连接策略测试器的"市场报价"（不是程序端）。

开始测试多货币EA之前，需要选择在程序端"市场报价"要求测试的交易品种和加载所需的数据。首次调用"foreign"交易品种期间，历史记录会在测试代理和客户端之间自动同步。"foreign"交易品种是不同于测试运行的交易品种。

推荐"other"交易品种的数据出现在下列情况下：

© 2000-2025, MetaQuotes Ltd.

- 当在交易品种/时间帧上使用技术指标函数 和 IndicatorCreate() 时；
- 请求其他交易品种的 "市场报价"数据：
  1. SeriesInfoInteger
  2. Bars
  3. SymbolSelect
  4. SymbolIsSynchronized
  5. SymbolInfoDouble
  6. SymbolInfoInteger
  7. SymbolInfoString
  8. SymbolInfoTick
  9. SymbolInfoSessionQuote
  10. SymbolInfoSessionTrade
  11. MarketBookAdd
  12. MarketBookGet
- 通过使用以下函数为交易品种/时间帧请求时间序列：
  1. CopyBuffer
  2. CopyRates
  3. CopyTime
  4. CopyOpen
  5. CopyHigh
  6. CopyLow
  7. CopyClose
  8. CopyTickVolume
  9. CopyRealVolume
  10. CopySpread

第一次调用 "other"交易品种时，测试进程停止，下载历史记录用于交易品种/时间帧，从程序端到测试代理。同时，生成该交易品种的订单号序列。

根据所选的订单号生成模式，为每个交易品种生成一个独立的订单号序列。您也可以通过调用OnInit()处理程序的SymbolSelect()明确请求期待的交易品种的历史记录 - 下载历史记录会在测试EA交易之前立即完成。

因此，在MetaTrader 5客户端无需任何额外的努力去执行多货币测试。只需在客户端打开相应交易品种的图表。如果包括该数据，那么所有所需交易品种的历史记录将会从交易服务器自动上传。

## 策略测试的模拟时间

测试期间，本地时间 TimeLocal() 始终等于服务器时间 TimeTradeServer()。反过来，服务器时间也始终等于相应GMT时间的时间 -TimeGMT()。这样，所有这些函数会在测试期间同时展示。

© 2000-2025, MetaQuotes Ltd.

如果没有连接服务器，由于GMT和本地时间没有差异，那么策略测试器中的服务器时间会刻意完成。无论连接与否，测试结果应该始终相同。有关服务器时间的信息也不会本地存储，而是来自服务器。

## 测试的图形对象

测试/优化期间图形对象无需标记。因此，当涉及测试/优化期间创建对象的属性时，EA交易将收到的是零值。

> 该限制不会应用于视觉模式的测试。

## 策略测试的OnTimer()函数

MQL5提供处理Timer事件的机会。无论何种测试模式都可以完成调用OnTimer()处理程序。这意味着如果在"只有开盘价"的模式，H4的周期下运行，并且EA有一个每秒调用的Timer设置，那么每个H4柱开始的时候，都将会调用一次OnTick()处理程序，调用14400次调用OnTimer()处理程序（3600秒*4小时）。EA的总测试时间将会根据EA的逻辑进行增加。

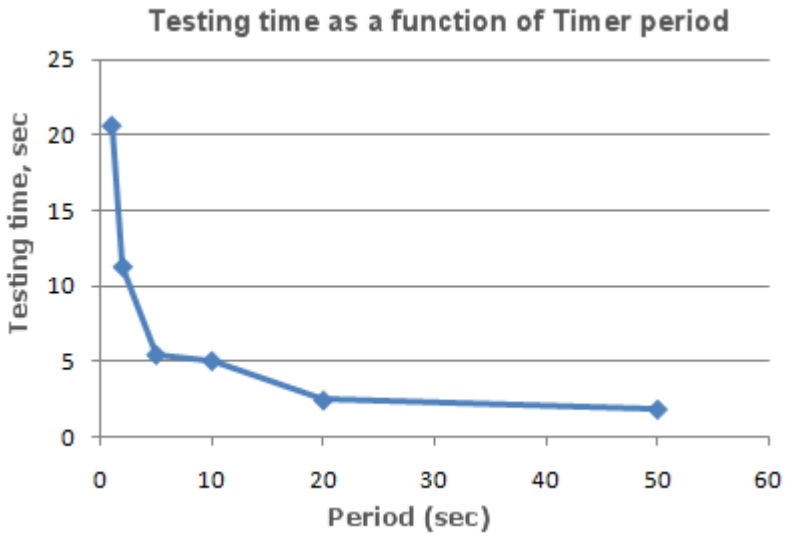
为了根据给予的Timer频率的测试时间进行检查，我们已经创建了无需任何交易操作的简单EA。

```
//--- 输入参数
input int      timer=1;               // Timer 值, 单位为秒
input bool     timer_switch_on=true; // 定时开关
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 如果timer_switch_on==true则运行 Timer
   if(timer_switch_on)
     {
      EventSetTimer(timer);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 停止 Timer
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| Timer 函数                                                       |
//+------------------------------------------------------------------+
void OnTimer()
```

© 2000-2025, MetaQuotes Ltd.

```
   {
//---
//  无操作，处理程序主体为空
   }
//+------------------------------------------------------------------+
```

在timer参数不同的值下，获得测试时间测量（Timer事件的周期性）。在获得的数据上，我们绘制测试时间作为Timer周期函数。.




可以很清晰的看到，越小的参数timer，优化EventSetTimer(Timer)函数期间，OnTimer()处理程序之间的调用越少，而同样情况下，测试时间T越大。

## 策略测试器中的Sleep() 函数

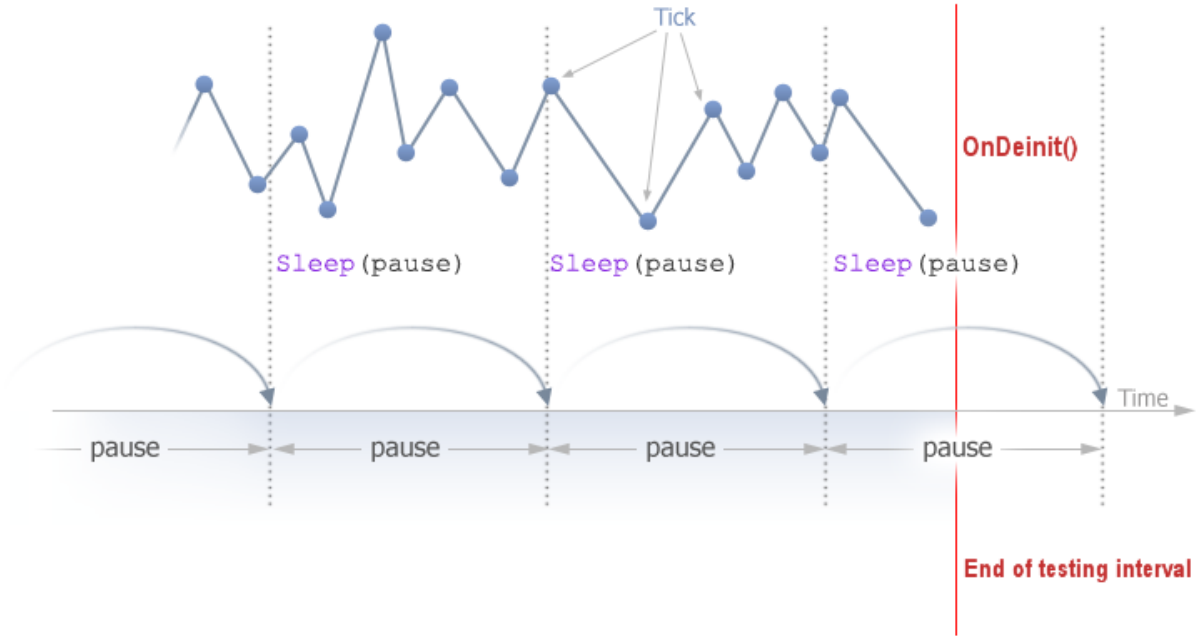
使用图形工作时，Sleep() 函数允许EA或脚本暂停执行mql5-程序。这在请求未准备并需要等待其准备的数据时非常有用。使用Sleep()函数的详细示例可以在数据访问安排部分找到。

Sleep()调用不会停滞测试过程。当您调用Sleep()时，生成的订单号会在指定延迟内“播放”，这可能会导致触发挂单，止损等等。Sleep()调用以后，策略测试器的模拟时间会增加间隔，这会在Sleep函数的参数中指明。

如果执行Sleep()函数的结果，策略测试器的当前时间检查测试时间，然后您将收到一个错误“测试时检测的无限睡眠循环”。如果您收到这个错误，测试结果会被驳回，在他们全交易量执行的所有计算（交易量，下降等），该测试的结果被传到程序端。

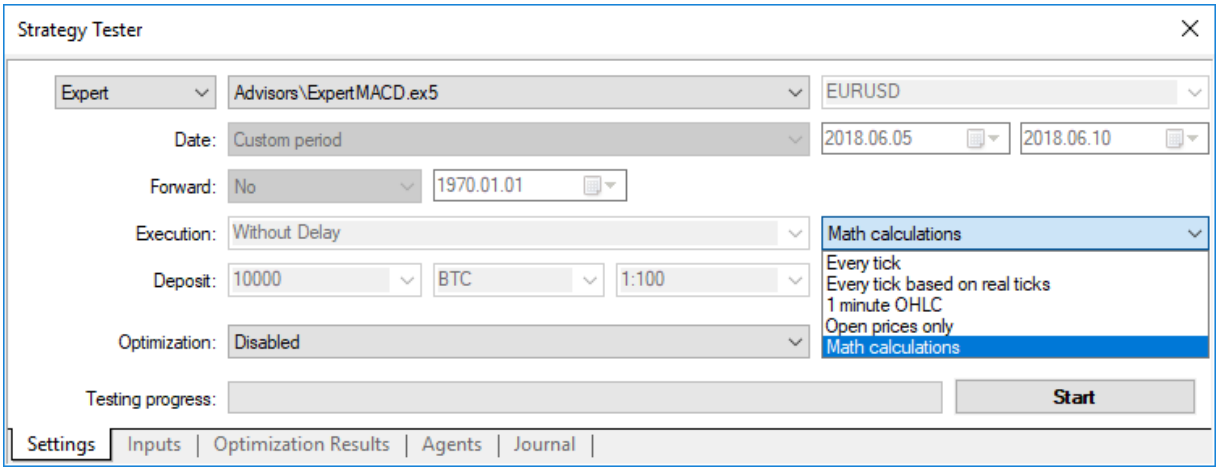
Sleep()函数在OnDeinit()不能工作，因为调用以后，测试时间将保证超过测试间隔范围。

© 2000-2025, MetaQuotes Ltd.




## 使用策略测试用于数学运算中的优化问题

使用MetaTrader 5 程序端的测试器，不仅能够测试交易策略，还能够用于数学运算。若要使用，必须选择 "数学运算"模式：




在这种情况下，将只有三种函数能被调用：OnInit()，OnTester()，OnDeinit()。在 "数学运算"模式，策略测试器不能生成任何订单号和下载历史记录。

如果您指定的开始日期大于结束日期，那么策略测试器也会以 "数学运算"的模式工作。

当使用测试器解决数学问题时，上传历史记录和生成订单号都不会发生。

在MetaTrader 5策略测试器用于解决的典型数学问题 - 搜索带有许多变量的函数极值。

要解决它，我们需要：

- 函数值的计算应该位于OnTester()函数；
- 函数参数必须被定义为EA交易的 导入变量 ；

© 2000-2025, MetaQuotes Ltd.

编译EA，打开 "策略测试器"的窗口。在 "导入参数"标签，选择所需的导入变量，通过为每个函数变量指定开始，停止和阶段值来定义参数值的设置。

选择优化类型 - "慢速完成算法"（全面搜索参数空间）或 "快速遗传基础算法"。为了简单搜索函数极值，最好选择快速优化，但是如果您想要计算整个变量设置的值，那么最好使用慢速优化。

选择 "数学运算"模式并使用 "开始"按钮，运行优化程序。注意优化时，策略测试器将搜索OnTester函数的最大值。若要找到本地最小值，则从OnTester函数返回相反的所计算的函数值：

```
return(1/function_value);
```

需要检查function_value不等于零，因为否则的话，我们可以获得除数为零的严重错误 。

还有另一种方式，它更方便并且不会曲解优化结果，这是这篇文章读者的建议：

```
return(-function_value);
```

该选项不需要检查等于零的function_value，3D表现法的优化结果表面有同样的形状，但是普通的镜像。

作为示例，我们提供了sink() 函数：

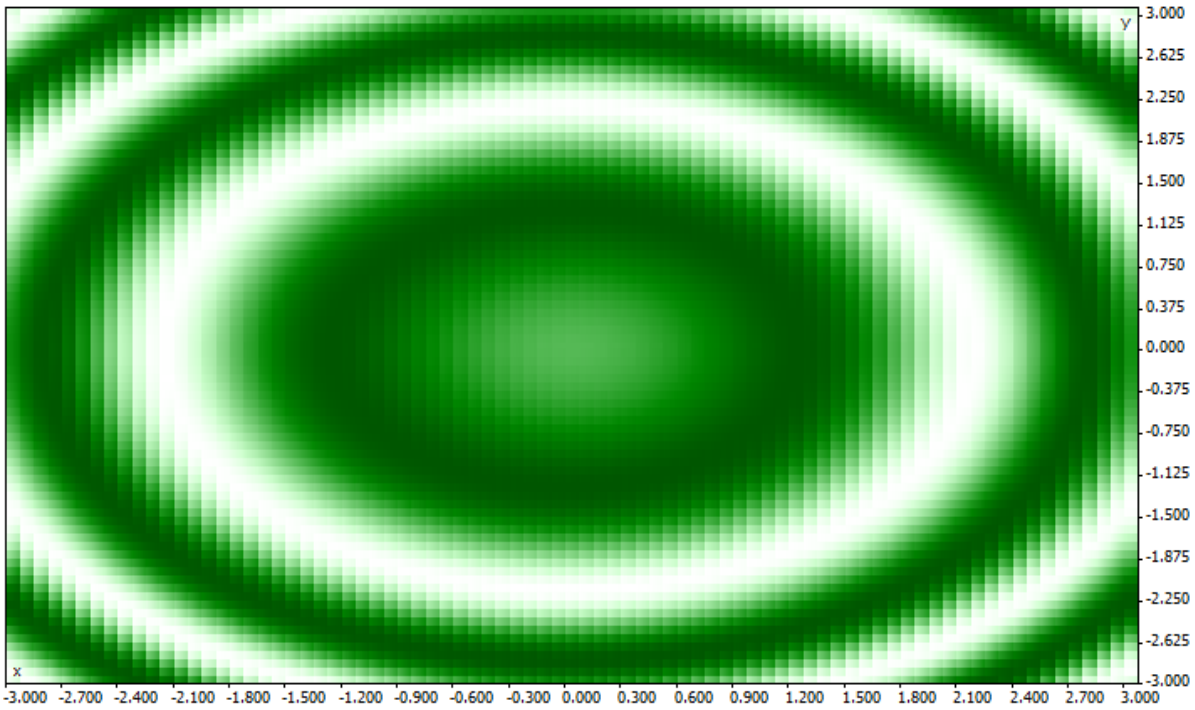
$$sink(x,y) = \sin(x^2 + y^2)$$

用于找到该函数极值的EA代码置于OnTester()：

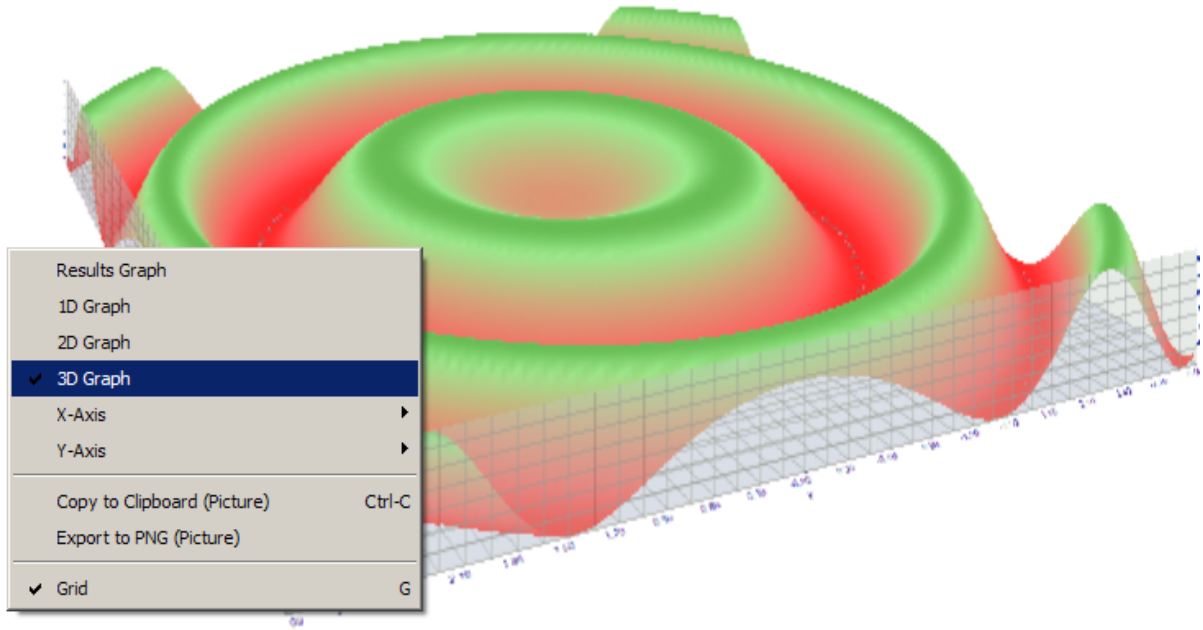
```
//+------------------------------------------------------------------+
//|                                                         Sink.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 输入参数
input double   x=-3.0; // start=-3, step=0.05, stop=3
input double   y=-3.0; // start=-3, step=0.05, stop=3
//+------------------------------------------------------------------+
//| Tester函数                                                       |
//+------------------------------------------------------------------+
double OnTester()
  {
//---
   double sink=MathSin(x*x+y*y);
//---
   return(sink);
  }
//+------------------------------------------------------------------+
```

执行优化并以2D图形的形式查看 优化结果 。

© 2000-2025, MetaQuotes Ltd.




给定参数组(x, y)的值越好，颜色饱和度越高。正如从查看sink()公式形式所期待的，它的值以中心为(0,0)形成同心圆。人们可以通过3D图形查看，sink()函数没有单独的全局极值：




## 在“只开盘价”模式同步柱形

MetaTrader 5 客户端的测试器允许我们检查所谓的“多货币”EA。多货币EA - 是交易两个或两个以上交易品种的EA。

在多交易品种上进行交易的策略测试，在测试器上强加了一些额外的技术要求：

- 为这些交易品种生成订单号；
- 为这些交易品种计算指标的值；

© 2000-2025, MetaQuotes Ltd.

- 为这些交易品种计算预付款规定额；
- 所有交易品种同步生成订单号序列。

按照所选的交易模式，策略测试器为每个工具生成和处理一个订单号序列。同时，打开每个交易品种的新柱，即使柱形在另一个交易品种打开。这意味着当多货币EA测试时，这种情况可能会产生（并经常产生），当一个工具时，一个新柱已经打开，而其他则不会。因此，测试时，一切都可能变为现实。

只要使用"每个订单号"和"1分钟OHLC"测试模式，测试器中历史记录的真实模拟就不会引起任何问题。对于这些模式来说，一个蜡烛图生成足够的订单号，以便能够等到同步来自不同交易品种的柱形。但是如果同步交易工具柱形是强制的，我们如何在"只开盘价"模式测试多货币策略？在该模式下，只在对应柱的开盘时间的一个订单号上调用EA。

我们将在一个示例上进行说明：如果我们在EA测试EURUSD，新的小时蜡烛图在EURUSD打开，那么我们可以很容易地辨认出这个事实 - "只开盘价"模式测试，事件NewTick 对应于测试周期的打开柱形时刻。但是不保证EA中使用的交易品种USDJPY打开的新蜡烛图。

正常情况下，它足以完成OnTick() 函数的工作和检查在下一个订单号出现的USDJPY新柱。但是当在"只开盘价"模式测试时，将不会有其他订单号，所以可能看起来这种模式不适合测试多货币EA。但是事实并非如此 - 不要忘了 MetaTrader 5 测试器的行为就如在现实生活一样。您可以等候直至使用Sleep()函数打开另一个交易品种的新柱！

EA Synchronize_Bars_Use_Sleep.mq5的代码，显示"只开盘价"模式的柱形同步示例：

```
//+------------------------------------------------------------------+
//|                                  Synchronize_Bars_Use_Sleep.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 输入参数
input string   other_symbol="USDJPY";
//+------------------------------------------------------------------+
//| EA初始化函数                                                     |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 检查交易品种
   if(_Symbol==other_symbol)
     {
      PrintFormat("You have to specify the other symbol in input parameters or select
      //--- 被迫停止测试
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA订单号函数                                                     |
```

```
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 静态变量，用于存储最近的柱形时间
   static datetime last_bar_time=0;
//--- sync 标识
   static bool synchonized=false;
//--- 如果静态变量未初始化
   if(last_bar_time==0)
     {
      //--- 这是第一次调用，节省柱形时间并退出
      last_bar_time=(datetime)SeriesInfoInteger(_Symbol,Period(),SERIES_LASTBAR_DATE);
      PrintFormat("The last_bar_time variable is initialized with value %s",TimeToStri
     }
//--- 获得图表交易品种最近柱的开盘时间
   datetime curr_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DAT
//--- 如果时间不平等
   if(curr_time!=last_bar_time)
     {
      //--- 保存柱形开盘时间到静态变量
      last_bar_time=curr_time;
      //--- 不同步
      synchonized=false;
      //--- 打印消息
      PrintFormat("A new bar has appeared on symbol %s at %s",_Symbol,TimeToString(Tir
     }
//--- 另一个交易品种柱形的开盘时间
   datetime other_time;
//--- 循环，直至另一个交易品种开盘时间等于curr_time
   while(!(curr_time==(other_time=(datetime)SeriesInfoInteger(other_symbol,Period(),SE
     {
      PrintFormat("Waiting 5 seconds..");
      //--- 等待 5 秒，调用SeriesInfoInteger(other_symbol,Period(),SERIES_LASTBAR_DATE)
      Sleep(5000);
     }
//--- 同步柱
   synchonized=true;
   PrintFormat("Open bar time of the chart symbol %s: is %s",_Symbol,TimeToString(last
   PrintFormat("Open bar time of the symbol %s: is %s",other_symbol,TimeToString(other
//--- TimeCurrent()无用，使用TimeTradeServer()
   Print("The bars are synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS))
  }
//+------------------------------------------------------------------+
```

注意，EA最近的行，显示建立同步事实的当前时间：

```
   Print("The bars synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS));
```

© 2000-2025, MetaQuotes Ltd.

要显示当前时间，我们使用 TimeTradeServer() 函数而非 TimeCurrent()。TimeCurrent()函数返回最近的订单号时间，该时间在使用Sleep()后不能改变。在 "只开盘价"模式运行EA，您将看到同步柱形的信息。

| | | |
|---|---|---|
| Core 1 | 2010.12.01 20:00:05 | The bars are synchronized at 2010.12.01 20:00:05 |
| Core 1 | 2010.12.01 20:00:05 | Open bar time of the chart symbol USDJPY: 2010.12.01 20:00 |
| Core 1 | 2010.12.01 20:00:05 | A new bar has appeared on symbol EURUSD: 2010.12.01 20:00 |
| Core 1 | 2010.12.01 20:00:00 | Waiting 5 seconds.. |
| Core 1 | 2010.12.01 20:00:05 | A new bar has appeared on symbol EURUSD: 2010.12.01 20:00 |
| Core 1 | 2010.12.01 16:00:05 | The bars are synchronized at 2010.12.01 16:00:05 |

如果您需要获得当前服务器时间，而不是最近订单号到达时间，请使用TimeTradeServer() 函数替代TimeCurrent()。

还有另一个同步柱形的方式 - 使用timer。这种EA的示例是Synchronize_Bars_Use_OnTimer.mq5，附于本文。

## 测试器的IndicatorRelease()函数

完成单项测试以后，工具图表自动打开，显示完成的交易和EA中使用的指标。这有助于视觉上检查进出点，并且与指标的值进行比较。

> 注意：在测试完成后自动打开的图表上显示的指标，会在测试完成后重新计算。即使这些指标应用在已测的EA上。

但是在某些情况下，程序员可能想要隐藏交易算法中包括的指标信息。例如，EA代码作为可执行文件被租用或出售，不提供源代码。为此，IndicatorRelease()函数更合适。

如果程序端以 tester.tpl的名称在客户端的directory/profiles/templates设置模板，那么它将被应用于打开的图表。如果没有，则会使用默认模板(default.tpl)。

如果不再需要，IndicatorRelease() 函数通常意于释放指标的计算部分。这可以使您节省内存和CPU两方面的资源，因为每个订单号调用都要计算指标。另一个目的 -就是单项测试运行后，禁止在测试图表显示指标。

若要在测试后，禁止在图表上显示指标，通过OnDeinit()处理程序调用IndicatorRelease() 来处理指标。完成后和显示测试图表前会始终调用OnDeinit()函数。

```
//+------------------------------------------------------------------+
//| EA去初始化函数                                                   |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   bool hidden=IndicatorRelease(handle_ind);
   if(hidden) Print("IndicatorRelease() successfully completed");
   else Print("IndicatorRelease() returned false. Error code ",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

若要在单项测试完成后禁止在图表上显示指标，请通过OnDeinit()处理程序使用函数IndicatorRelease()。

## 测试器中的事件处理

The presence of the EA的OnTick()处理程序并不是强制性的，要在MetaTrader 5测试器测试历史数据。EA中至少包括一个以下函数就已经足够：

- OnTick() - 到达新订单号的事件处理程序；
- OnTrade() - 交易事件处理程序；
- OnTimer() - timer信号到达的事件处理程序；
- OnChartEvent() - 客户事件处理程序。

在EA测试时，我们可以使用OnChartEvent()函数处理自定义事件，但是在指标中，该函数不能在测试器中调用。即使指标拥有OnChartEvent()事件处理程序并且该指标用在测试的EA，指标本身仍然不能接收任何自定义事件。

测试期间，指标可以使用EventChartCustom()函数生成自定义事件，并且EA可以用OnChartEvent()处理该事件。

除了这些事件，与测试和优化相关的特殊事件也在策略测试器生成。

- Tester - 该事件在完成EA历史数据测试后生成。Tester事件使用OnTester()函数处理。该函数仅在测试EA时被使用，主要意于计算用作自定义极大极小标准的值，用于导入参数的遗传优化。
- TesterInit - 该事件在第一次通过之前在策略测试器优化开始时生成。TesterInit事件使用OnTesterInit()函数处理。优化开始期间，该处理程序的EA交易自动加载于测试器指定交易品种和周期的独立程序端图表上，接收TesterInit事件。该函数用于在优化开始之前启动EA交易，以便进一步处理优化结果。
- TesterPass - 该事件在接收新数据框时生成。TesterPass事件使用OnTesterPass()函数处理。优化期间，该处理程序的EA交易自动加载于测试器指定交易品种和周期的独立程序端图表上，并在收到框架时接收TesterPass事件。该函数用于"立时"动态处理优化结果无需等候完成。使用FrameAdd()函数添加框架，该函数可以在单次通过结束后用OnTester()处理程序调用。
- TesterDeinit - 该事件在结束策略测试器的EA交易优化后生成。TesterDeinit事件使用OnTesterDeinit()函数处理。该处理程序的EA交易在优化开始时自动加载于图表，而在完成后接收TesterDeinit。该函数用于最后处理所有优化结果。

## 测试代理

MetaTrader 5客户端测试使用测试代理实施。自动创建和启用本地代理。默认的本地对象数量等于计算机的内核数量。

每个测试代理都有其自己的全局变量副本，它与客户端无关。程序端本身是调度员，将任务分配给本地和远程代理。执行一个测试EA任务以后，使用给定的参数，代理返回结果到程序端。对于单项测试，只使用一个代理。

代理存储从程序端接收的历史记录，根据工具名称，存储在独立的文件夹，所以EURUSD历史记录存储在名为EURUSD的文件夹。另外，工具历史记录也根据其来源分别处理。存储历史记录的结构看起来像以下方式：

```
tester_catalog\Agent-IPaddress-Port\bases\name_source\history\symbol_name
```

© 2000-2025, MetaQuotes Ltd.

例如，服务器MetaQuotes-Demo的EURUSD历史记录可以被存在文件夹tester_catalog\Agent-127.0.0.1-3000\bases\MetaQuotes-Demo\EURUSD。

完成测试后，本地代理，进入待机模式，等候5分钟，等待下一个任务，以便不在启动下次调用上浪费时间。只有等候期结束后，本地代理才会关闭，释放所占用的CPU内存。

如果提前完成测试，从用户方面（"取消"按钮），以及关闭客户端，所有本地代理会立即停止工作，释放所占用的内存。

## 程序端和代理之间的数据交换

当您运行测试时，客户端已准备向代理发送一些的参数模块：

- 用于测试的导入参数（模拟模式，测试间隔，工具，优化准则，等等）
- 选定的"市场报价"交易品种列表
- 交易品种测试规则（合约大小，获准的市场预付款用于设置止损和获利，等等）
- 要被测试的EA交易和其导入参数的值
- 附加文件信息（程序库，指标，数据文件 - # property tester_ ...）

| | | |
|---|---|---|
| tester_indicator | string | "indicator_name.ex5"格式的自定义指标名称。如果对应的参数通过常量字符串设置，那么需要测试的指标通过调用iCustom()函数自动定义。针对所有其他情况（使用IndicatorCreate()函数或使用设置指标名称的参数中的非常量字符串）需要该属性 |
| tester_file | string | 带有扩展指证的测试器文件名，双引号（常量字符串）。指定文件将被传到测试器。如果有必要，要被测试的导入文件，必须始终指明。 |
| tester_library | string | 带有扩展名的程序库名称，双引号。程序库可以有dll或ex5扩展名。需要测试的程序库被自动定义。然而，如果用自定义指标使用任何程序库，则需要该属性。 |

每个参数模块，都以MD5-hash的形式创建数字指纹，它会被发送至代理。MD5-hash每次设置都是独一无二的，其交易量比计算的信息数量少很多倍。

代理接收散列模块，将其与现有的进行比较。如果代理中没有给定的参数模块指纹，或接收的散列模块不同于现有的，那么代理请求该模块参数。这会减少程序端和代理之间的流量。

测试之后，代理返回程序端，运行的所有结果显示在"测试结果"和"优化结果"标签：所获利润，成交量，夏普系数，OnTester()函数的结果，等等。

优化期间，程序端以小包形式分发测试任务给代理，每包都包含几个任务（每个任务意味着一系列导入参数的单独测试）。这减少了程序端和代理之间的交换时间。

出于安全原因考虑，从程序端（EA，指标，程序库等等）获得的EX5文件，代理从不将其记录到硬盘，以便于运行代理的计算机不使用发送的数据。所有其他文件，包括DLL，记录在沙箱。在远程代理，您不能使用DLL测试EA。

测试结果通过程序端添加到特殊的结果缓存（结果缓存），以便使用时可以快速访问。对于每个参数设置，程序端都会搜索结果缓存，获得之前运行提供的结果，为了避免重新运行。如果无法找到这种参数的结果，代理会被给予任务引导测试。

© 2000-2025, MetaQuotes Ltd.

程序端和代理之间的所有流量加密。

> 订单号并不是通过网络发送，它们在测试生成代理。

## 使用所有客户端的共享文件夹

所有测试代理都来自客户端，相互独立：每个代理都有其自己的日志记录文件夹。另外，代理测试期间的所有操作都发生在**agent_name/MQL5/Files**文件夹。然而，如果打开文件期间您指定标识FILE_COMMON，那么我们可以通过客户端的共享文件夹实施本地代理和客户端的互动：

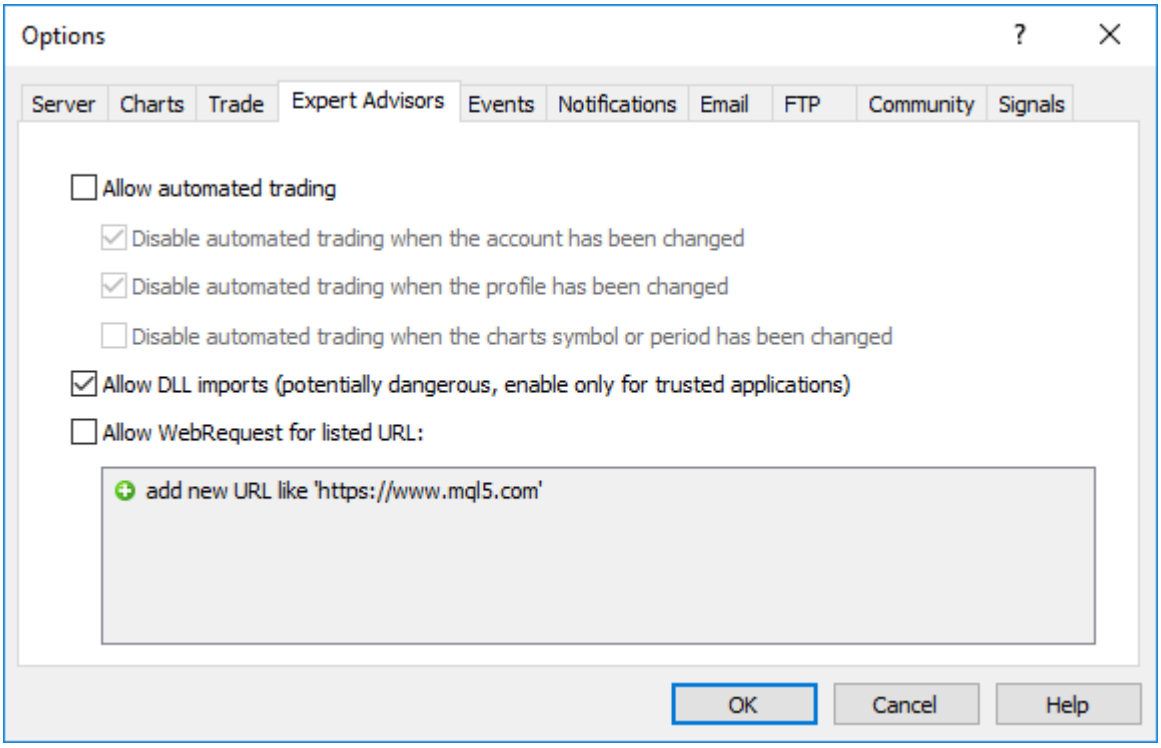
```
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 所有客户端的共享文件夹
   common_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- 拟定该文件夹的名称
   PrintFormat("Open the file in the shared folder of the client terminals %s", commor
//--- 打开共享文件夹的文件 FILE_COMMON flag指定)
   handle=FileOpen(filename,FILE_WRITE|FILE_READ|FILE_COMMON);
   ... further actions
//---
   return(INIT_SUCCEEDED);
  }
```

## 使用DLLs

所要加速优化，我们不仅可以使用本地，还可以使用 远程代理。这种情况下，对远程代理会有一些限制。首先，在日志中远程代理不显示Print()函数的执行结果，持仓和平仓的信息。日志中仅显示最少的信息，来防止在远程代理工作的电脑上，错误地编写EA。

另一个限制 - 测试EA时禁止使用DLL。出于安全考虑，远程代理完全禁止调用DLL。在本地代理，也只在相应的“允许导入DLL”权限下，允许在测试EA中调用DLL。

© 2000-2025, MetaQuotes Ltd.




注意： 当使用从EA (脚本，指标)接收的需要允许DLL调用时，您应该意识到在程序端设置中允许该选项时您要承担的风险。无论EA如何使用 - 是用于测试还是运行图表。

© 2000-2025, MetaQuotes Ltd.

# 预定义变量

对于每个可执行的MQL5程序的一套变量都是支持的，可以通过MQL5程序反应当前即时价格（EA交易、脚本或者自定义指标）。

预先定义变量的值在MQL5程序启动之前通过客户端建立，预先定义变量是不变的，也不会通过MQL5程序转变。例外是，特殊变量_LastError，通过 ResetLastError 函数重设至0

| 变量 | 值 |
| --- | --- |
| _AppliedTo | _AppliedTo变量可以找出用于指标计算的数据类型 |
| _Digits | 小数位数字 |
| _Point | 在引用货币中当前交易品种的大小 |
| _LastError | 最后的错误代码 |
| _Period | 当前表格的时间表 |
| _RandomSeed | 假随机整数生成器的当前状态 |
| _StopFlag | 停止标记程序 |
| _Symbol | 当前图表的交易品种名称 |
| _UninitReason | 阻止初始化原因代码 |
| _IsX64 | _IsX64变量可以找出运行MQL5应用程序的程序端的位元版本 |

数据库被程序变量调用使用。

© 2000-2025, MetaQuotes Ltd.

# int _AppliedTo

_AppliedTo变量可以找出用于指标计算的数据类型：

| 数据类型 | 含义 | 用于指标计算的数据描述。 |
|---|---|---|
| — | 0 | 这个指标使用第二个OnCalculate()调用表格 - 计算的数据不由某个缓冲区或数据数组指定 |
| 关闭 | 1 | 收盘价 |
| 打开 | 2 | 开盘价 |
| 高级别 | 3 | 最高价 |
| 低级别 | 4 | 最低价 |
| 中间价(HL/2) | 5 | 中间价 = (High+Low)/2 |
| 典型价格(HLC/3) | 6 | 典型价格 = (High+Low+Close)/3 |
| 加权价格(HLCC/4) | 7 | 加权价格= (Open+High+Low+Close)/4 |
| 之前指标数据 | 8 | 该指标之前，图表上启用的指标数据。 |
| 第一个指标数据 | 9 | 图表上最先启用的指标数据 |
| 指标句柄 | 10+ | 使用指标句柄传递到iCustom()函数的指标数据。_AppliedTo值包含指标句柄 |

例如：

```
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲映射
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
// 获得用于指标计算的数据类型
   Print("_AppliedTo=",_AppliedTo);
   Print(getIndicatorDataDescription(_AppliedTo));
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 用于指标计算的数据描述                                           |
//+------------------------------------------------------------------+
string getIndicatorDataDescription(int data_id)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   string descr="";
   switch(data_id)
     {
      case(0):descr="It's first type of OnCalculate() - no data buffer";
         break;
      case(1):descr="Indicator calculates on Close price";
         break;
      case(2):descr="Indicator calculates on Open price";
         break;
      case(3):descr="Indicator calculates on High price";
         break;
      case(4):descr="Indicator calculates on Low price";
         break;
      case(5):descr="Indicator calculates on Median Price (HL/2)";
         break;
      case(6):descr="Indicator calculates on Typical Price (HLC/3)";
         break;
      case(7):descr="Indicator calculates on Weighted Price (HLCC/4)";
         break;
      case(8):descr="Indicator calculates Previous Indicator's data";
         break;
      case(9):descr="Indicator calculates on First Indicator's data";
         break;
      default: descr="Indicator calculates on data of indicator with handle="+string(c
         break;
     }
//---
   return descr;
  }
```

另见

ENUM_APPLIED_PRICE

© 2000-2025, MetaQuotes Ltd.

# int _Digits

_Digits变量保留小数点后几位，定义当前图表交易品种的价格精确度

可以使用 Digits() 函数。

© 2000-2025, MetaQuotes Ltd.

# double _Point

_Point变量包括货币报价中当前交易品种的大小点

可以使用 Point() 函数。

© 2000-2025, MetaQuotes Ltd.

# int _LastError

_LastError 变量包括最后 错误 代码，在MQL5程序运行中发生，使用 ResetLastError() 值能重设成0。

获得最后错误代码，使用 GetLastError() 函数。

© 2000-2025, MetaQuotes Ltd.

# ENUM_TIMEFRAMES _Period

_Period变量包括当前图表时间表的值。

也可以使用 Period() 函数。

另见

PeriodSeconds, 图表时间表, 日期和时间, 对象可见性

© 2000-2025, MetaQuotes Ltd.

# _RandomSeed

生成假随机整数时存储当前状态的变量。 调用 MathRand()时改变_RandomSeed 的值。使用 MathSrand() 设置所需的初始条件。

**x** 通过MathRand()函数接收的随机号在每次调用时按照以下方式计算：

```
x=_RandomSeed*214013+2531011;
_RandomSeed=x;
x=(x>>16)&0x7FFF;
```

另见

MathRand(), MathSrand(), Integer types

© 2000-2025, MetaQuotes Ltd.

# int _StopFlag

_StopFlag变量包含mql5程序停止标识，并且在正常运行期间等于0。当客户端尝试停止程序时，该变量将被设置为非零值。

为了检测_StopFlag，你可以使用 IsStopped 函数。

© 2000-2025, MetaQuotes Ltd.

## string _Symbol

_Symbol变量包括当前图表的交易品种名称。

也可以使用 Symbol() 函数。

© 2000-2025, MetaQuotes Ltd.

# int _UninitReason

_UninitReason变量包括 无法初始化程序中的代码。

通常，UninitializeReason()函数中包括代码。

© 2000-2025, MetaQuotes Ltd.

# int _IsX64

_IsX64变量可以找出运行MQL5应用程序的程序端的位元版本：_IsX64=0用于32位程序端，_IsX64!=0 用于64位程序端。

此外，还可以使用TerminalInfoInteger(TERMINAL_X64)函数。

例如：

```
// 检查程序运行的程序端
   Print("_IsX64=",_IsX64);
   if(_IsX64)
      Print("Program ",__FILE__," is running in the 64-bit terminal");
   else
      Print("Program ",__FILE__," is running in the 32-bit terminal");
   Print("TerminalInfoInteger(TERMINAL_X64)=",TerminalInfoInteger(TERMINAL_X64));
```

另见

MQLInfoInteger, 输入函数 (#import)

© 2000-2025, MetaQuotes Ltd.

# 普通函数

专业集合里不包括的一般用途函数如下表。

| 函数 | 命令 |
| --- | --- |
| Alert | 在独立窗口中显示消息 |
| CheckPointer | 返回目标指针类型 |
| Comment | 图表左上角输出注解文本 |
| CryptEncode | 通过指定方式变换来自数组的数据 |
| CryptDecode | 执行数组数据逆转换 |
| DebugBreak | 排除故障中的程序断点 |
| ExpertRemove | 停止智能交易并从图表中卸载 |
| GetPointer | 返回对象 指针 |
| GetTickCount | 从系统开始，返回已经过去的毫秒的数量 |
| GetTickCount64 | 从系统开始，返回已经过去的毫秒的数量 |
| GetMicrosecondCount | 返回MQL5程序开始后过去的微秒数 |
| MessageBox | 创建，显示消息盒子，然后管理 |
| PeriodSeconds | 在周期中返回秒钟数量 |
| PlaySound | 播放第二个文件 |
| Print | 显示未经处理的消息 |
| PrintFormat | 在目标文件中按照原来格式，格式刷建立的交易品种和值 |
| ResetLastError | 建立预定义变量 _LastError的值到0 |
| ResourceCreate | 基于数据设置创建图像资源 |
| ResourceFree | 删除 动态创建资源 (释放分配的内存) |
| ResourceReadImage | 阅读来自通过 ResourceCreate() 函数创建的 或编译期间保存在EX5文件的图形资源的数据 |
| ResourceSave | 将资源保存在指定文件 |
| SetUserError | 创建预先参量 _LastError，使其值等于 ERR_USER_ERROR_FIRST + user_error |
| SetReturnError | 设置完成操作时返回程序端进程的代码 |
| Sleep | 延迟执行当前EA交易或者脚本的规定间隔 |
| TerminalClose | 命令客户端完成操作 |
| TesterHideIndicators | 设置显示/隐藏EA中使用的指标的模式 |
| TesterStatistics | 返回基于测试结果计算的指定统计的值 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 命令 |
| --- | --- |
| TesterStop | 当测试时，给予程序操作完成命令 |
| TesterDeposit | 在测试期间模拟存款的特殊函数。它在一些资金管理系统中使用。 |
| TesterWithdrawal | 在测试过程中特殊函数模仿取消内存操作，可以用于资产管理系统中 |
| TranslateKey | 通过考虑到当前输入语言和控制键状态的虚拟键码返回Unicode字符 |
| ZeroMemory | 通过引用重设变量传递，该变量可以是任何类型，除了分类和结构都有构造函数 |

© 2000-2025, MetaQuotes Ltd.

# Alert

在单独窗口中显示信息。

```
void  Alert(
   argument,      // 第一值
   ...            // 其他值
   );
```

参量

*argument*

[in] 任意值，被逗号分开，分开信息输出分类量，可以使用线路供应字符"\n"或者 "\r\n". 参量的数量不能超过64个

返回值

没有返回值。

注释

数组不能通过Alert() 函数传递，数组应该输出元素，双精度类型数据可以输出小数点后8位以上，浮点类型数据可以输出小数点后5位，不同精度起源不同数字或者系统模式，使用 DoubleToString()。

布尔型数据以true或者false字符串输出，日期以 YYYY.MM.DD HH:MI:SS格式，使用 TimeToString()日期另外模式显示日期，颜色类型数据既可以RGB字符串或者颜色名称输出，颜色存在颜色设置中。

Alert() 函数在策略测试中不工作。

示例：

```
//--- 枚举
enum ENUM_INTERSECT_DIRECTION
  {
   INTERSECT_DIRECTION_NONE= 0,  // no crossing
   INTERSECT_DIRECTION_UP  = 1,  // upward crossing
   INTERSECT_DIRECTION_DOWN=-1,  // downward crossing
  };

//--- 输入参数
input    uint                InpPeriod = 10;            // MA Period
input    int                 InpShift  = 0;             // MA Shift
input    ENUM_MA_METHOD      InpMethod = MODE_SMA;      // MA Method
input    ENUM_APPLIED_PRICE  InpPrice  = PRICE_CLOSE;   // MA Applied price

//--- 全局变量
int      ExtMaHandle;
int      ExtMaPeriod;
double   ExtData[2];
```

© 2000-2025, MetaQuotes Ltd.

```
MqlRates ExtRates[2];

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 如果输入参数中指定为零，则计算移动平均线的周期将等于默认值(10)
   ExtMaPeriod=int(InpPeriod<1 ? 10 : InpPeriod);
//--- 用指定的参数创建移动平均指标的句柄
   ExtMaHandle=iMA(Symbol(),PERIOD_CURRENT,ExtMaPeriod,InpShift,InpMethod,InpPrice);
   ResetLastError();
   if(ExtMaHandle==INVALID_HANDLE)
     {
      PrintFormat("Failed to create iMA() handle. Error code: %d",GetLastError());
      return(INIT_FAILED);
     }

//--- 获取最后价更新的时间
   datetime tick_time=TickTime();
//--- 获取最后两个柱形图的移动平均线数据和价格数据
   if(GetData(ExtMaHandle,ExtData,ExtRates) && tick_time!=0)
     {
      //--- 如果价格高于移动平均线(MA)
      if(ExtRates[1].close>ExtData[1])
        {
         //--- 创建消息文本并显示预警(Alert)
         string message=StringFormat("Bar time: %s. The price is above the moving aver
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECON
         /*
        结果:
         Alert: Bar time: 2024.02.16 18:00. The price is above the moving average at 2
         */
        }
      else
        {
         //--- 如果价格低于移动平均线(MA)
         if(ExtRates[1].close<ExtData[1])
           {
            //--- 创建消息文本并显示预警(Alert)
            string message=StringFormat("Bar time: %s. The price is below the moving a
            Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SE
            /*
           结果:
            Alert: Bar time: 2024.02.16 19:00. The price is below the moving average a
            */
           }
         else
           {
```

© 2000-2025, MetaQuotes Ltd.

```
            //--- 创建消息文本并显示预警(Alert)
            string message=StringFormat("Bar time: %s. The price and moving average ar
            Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SE
            /*
           结果:
            Alert: Bar time: 2024.02.16 20:00. The price and moving average are equal
            */
           }
        }
     }

//--- 成功
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   ResetLastError();
//--- 获取最后两个柱形图的移动平均线数据和价格数据
   if(!GetData(ExtMaHandle,ExtData,ExtRates))
      return;
//--- 在当前柱形图获取价格穿过移动平均线的方向
   ENUM_INTERSECT_DIRECTION intersect=GetIntersectDirection(ExtData,ExtRates);

//--- 用于保存前一条消息的变量
   static string message_prev="";

//--- 如果价格在当前柱形图向上穿过移动平均线
   if(intersect==INTERSECT_DIRECTION_UP)
     {
      //--- 获取穿过时的报价时间
      datetime tick_time=TickTime();
      if(tick_time==0)
         return;
      //--- 创建消息文本
      string message=StringFormat("Bar time: %s. The price crossed the MA from bottom
      //--- 如果前一条消息与当前消息不一致，则显示预警(Alert)，包含消息和报价时间
      if(message!=message_prev)
        {
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECON
         message_prev=message;
         /*
         Result:\
         Alert: Bar time: 2024.02.16 09:00. The price crossed the MA from bottom to to
         */
        }
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果价格在当前柱形图向下穿过移动平均线
   if(intersect==INTERSECT_DIRECTION_DOWN)
     {
      //--- 获取穿过时的报价时间
      datetime tick_time=TickTime();
      if(tick_time==0)
         return;
      //--- 创建消息文本
      string message=StringFormat("Bar time: %s. The price crossed the MA from top to
      //--- 如果前一条消息与当前消息不一致，则显示预警(Alert)，包含消息和报价时间
      if(message!=message_prev)
        {
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECON
         message_prev=message;
         /*
         Result:\
         Alert: Bar time: 2024.02.16 10:00. The price crossed the MA from top to botto
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| 将价格和移动平均线数据放入数组                                   |
//+------------------------------------------------------------------+
bool GetData(int handle,double &ma_data[],MqlRates &price_data[])
  {
   ResetLastError();
//--- 获取最后两个柱形图的移动平均线数据
   if(CopyBuffer(handle,0,0,2,ma_data)!=2)
     {
      PrintFormat("CopyBuffer() failed. Error code: %d",GetLastError());
      return(false);
     }
//--- 获取最后两个柱形图的价格数据
   if(CopyRates(Symbol(),PERIOD_CURRENT,0,2,price_data)!=2)
     {
      PrintFormat("CopyRates() failed. Error code: %d",GetLastError());
      return(false);
     }

   return(true);
  }
//+------------------------------------------------------------------+
//| 返回价格穿过移动平均线的方向                                     |
//+------------------------------------------------------------------+
ENUM_INTERSECT_DIRECTION GetIntersectDirection(double &ma_data[],MqlRates &price_data
  {
   double ma0=ma_data[1];
```

© 2000-2025, MetaQuotes Ltd.

```
   double ma1=ma_data[0];
   double close0=price_data[1].close;
   double close1=price_data[0].close;

   if(close1<=ma1 && close0>ma0)
      return(INTERSECT_DIRECTION_UP);
   else
     {
      if(close1>=ma1 && close0<ma0)
         return(INTERSECT_DIRECTION_DOWN);
      else
         return(INTERSECT_DIRECTION_NONE);
     }
  }
//+------------------------------------------------------------------+
//| 返回以秒为单位的报价时间                                         |
//+------------------------------------------------------------------+
datetime TickTime()
  {
   MqlTick tick={};

   ResetLastError();
   if(!SymbolInfoTick(Symbol(),tick))
     {
      PrintFormat("SymbolInfoTick() failed. Error code: %d",GetLastError());
      return(0);
     }

   return(tick.time);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckPointer

函数返回 指针对象类型

```
ENUM_POINTER_TYPE  CheckPointer(
   object* anyobject      // 对象指针
   );
```

参量

*anyobject*

[in] 对象指针。

返回值

从 ENUM_POINTER_TYPE 枚举中返回值。

注释

想要调用不正确的指针到危险终止结果程序中，在使用指针之前，调用CheckPointer函数是有必要的，如下条件下，指针是不正确的：

- 指针等于 NULL;
- 使用 delete 操作删除对象；

该函数可以用来正确检测指针，非零值警告指针在访问时使用。

要快速验证指针，您还可以使用操作符"!" (示例)通过隐式调用CheckPointer函数来检查。

示例：

```
//+------------------------------------------------------------------+
//| 通过删除元素来删除列表                                           |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- 循环工作服务指针
   CItem* item;
//--- 循环检测并删除动态指针
   while(CheckPointer(m_items)!=POINTER_INVALID)
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
        {
         Print("Dynamic object ",item.Identifier()," to be deleted");
         delete (item);
        }
      else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
     }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

另见

对象指针，检测对象指针，对象删除操作符的删除

© 2000-2025, MetaQuotes Ltd.

# Comment

图表左上角函数通过用户输出注解。

```
void  Comment(
   argument,      // 第一值
   ...            // 下一值
   );
```

参量

...

[in]任何值，逗号分隔。划线输出信息到几条路线，使用一条换行符 "\n" 或者 "\r\n"，参量数不能超过64个。输入注解文本 (包括隐形交易品种)的总长度不能超过2045个字节 (在输出过程中超出的交易品种会被删去)

返回值

没有返回值

注释

数组不能通过Comment()函数传递，数组必须一个接一个元素进入。

双精度型数组的精确度保留小数点后16位，以传统或者科学的方式输出，以保证接入密切。浮点型数据保留小数点后5位，为了以另一精确度或者预定义模式输出数字，使用 DoubleToString() 函数。

布尔型数据以true或者false输出，数据显示成 YYYY.MM.DD HH:MI:SS.显示另一模式数据，使用 TimeToString(). 颜色型数据以RGB或者颜色名称显示，如果该颜色在颜色设置中存在。

Comment() 函数在优化期间在策略测试中不工作。

示例：

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
//--- 3行输出值
   Comment(StringFormat("Show prices\nAsk = %G\nBid = %G\nSpread = %d",Ask,Bid,Spread)
  }
```

另见

ChartSetString, ChartGetString

© 2000-2025, MetaQuotes Ltd.

# CryptEncode

通过指定方式转换来自数组的数据。

```
int  CryptEncode(
   ENUM_CRYPT_METHOD   method,        // 方法
   const uchar&        data[],        // 源数组
   const uchar&        key[],         // 密钥
   uchar&              result[]       // 目标数组
   );
```

参数

*method*

[in] 数据转换方法。可以是ENUM_CRYPT_METHOD枚举的一个值。

*data[]*

[in] 源数组。

*key[]*

[in] 密钥数组。

*result[]*

[out] 目标数组。

返回值

错误情况下返回目标数组的字节数或0。若要获得有关错误 的信息，调用 GetLastError() 函数。

例如：

```
//+------------------------------------------------------------------+
//| ArrayToHex                                                       |
//+------------------------------------------------------------------+
string ArrayToHex(uchar &arr[],int count=-1)
  {
   string res="";
//--- 检查
   if(count<0 || count>ArraySize(arr))
      count=ArraySize(arr);
//--- 转换到 HEX 字符串
   for(int i=0; i<count; i++)
      res+=StringFormat("%.2X",arr[i]);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   string text="The quick brown fox jumps over the lazy dog";
   string keystr="ABCDEFG";
   uchar src[],dst[],key[];
//--- 准备密钥
   StringToCharArray(keystr,key);
//--- 复制文本到源数组src[]
   StringToCharArray(text,src);
//--- 打印初始数据
   PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(s
//--- 用key[]的DES56位密钥加密DES src[]
   int res=CryptEncode(CRYPT_DES,src,key,dst);
//--- 检查错误
   if(res>0)
     {
      //--- 打印加密数据
      PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
      //--- 解码 dst[] 到 src[]
      res=CryptDecode(CRYPT_DES,dst,key,src);
      //--- 检查错误
      if(res>0)
        {
         //--- 打印解码数据
         PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToSt
        }
      else
         Print("Error in CryptDecode. Error code=",GetLastError());
     }
   else
      Print("Error in CryptEncode. Error code=",GetLastError());
  }
```

另见

数组函数, CryptDecode()

© 2000-2025, MetaQuotes Ltd.

# CryptDecode

执行数组数据逆转换，通过CryptEncode()转换。

```
int  CryptEncode(
   ENUM_CRYPT_METHOD   method,        // 方法
   const uchar&        data[],        // 源数组
   const uchar&        key[],         // 密钥
   uchar&              result[]       // 目标数组
   );
```

参数

*method*

[in] 数组转换方法。可以是ENUM_CRYPT_METHOD 枚举的一个值。

*data[]*

[in] 源数组。

*key[]*

[in] 密钥数组。

*result[]*

[out] 目标数组。

返回值

错误情况下返回目标数组的字节数或0。若要获得有关错误 信息，调用 GetLastError() 函数。

示例：

```
input string InpKey = "ABCDEFG";  // 加密密钥

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string text="The quick brown fox jumps over the lazy dog";
   uchar src[],dst[],key[];

//--- 准备加密密钥
   StringToCharArray(InpKey,key);
//--- 准备src[]源数组
   StringToCharArray(text,src);
//--- 显示源数据
   PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(s

//--- 使用DES方法加密src[]数组，key[]为56位密钥
   int res=CryptEncode(CRYPT_DES,src,key,dst);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 检查加密结果
   if(res>0)
     {
      //--- 显示加密数据
      PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
      //--- 使用DES方法解密dst[]数组数据,key[]为56位密钥
      res=CryptDecode(CRYPT_DES,dst,key,src);
      //--- 检查结果
      if(res>0)
        {
         //--- 显示解密数据
         PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToSt
        }
      else
         Print("CryptDecode failed. Error: ",GetLastError());
     }
   else
      Print("CryptEncode failed. Error: ",GetLastError());
  }
//+------------------------------------------------------------------+
//| ArrayToHex                                                       |
//+------------------------------------------------------------------+
string ArrayToHex(uchar &arr[],int count=-1)
  {
   string res="";

//--- 检查大小
   if(count<0 || count>ArraySize(arr))
      count=ArraySize(arr);

//--- 转到hexadecimal字符串
   for(int i=0; i<count; i++)
      res+=StringFormat("%.2X",arr[i]);

   return(res);
  }
```

另见

数组函数, CryptEncode()

© 2000-2025, MetaQuotes Ltd.

# DebugBreak

在排除故障中的程序断点。

```
void  DebugBreak();
```

返回值

没有返回值。

注释

如果程序以调试的方式打开，MQL5程序的执行被打断。函数可以使用变量的观察值和/或一步步执行。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 使用F5编译文件
//--- 在调试模式下，如果i == j，则在DebugBreak()字符串停止
   for(int i=0,j=20; i<20; i++,j--)
     {
      if(i==j)
         DebugBreak();
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# ExpertRemove

函数停止EA交易并从图表中卸载。

```
void  ExpertRemove();
```

返回值

没有返回值。

注释

一旦调用ExpertRemove()函数，EA交易不会立即停止，设立一个停止EA操作的标志，任何事件都不能处理，调用 OnDeinit()，然后EA交易就会卸载并且从图表中删除。

在OnInit()处理程序内的策略测试中调用ExpertRemove()取消对当前参数集的测试。这种完成被认为是初始化错误。

当EA成功初始化之后在策略测试调用ExpertRemove()时，测试通常通过调用OnDeinit()和OnTester()来正常完成。在这种情况下，获取整个交易统计数据和优化准则的值。

示例：

```
//+------------------------------------------------------------------+
//|                                            Test_ExpertRemove.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20;//  EA 卸载前的订单号
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(TimeCurrent(),": " ,__FUNCTION__," reason code = ",reason);
//--- ”清除” 注释
   Comment("");
//---
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                     |
//+------------------------------------------------------------------+
void OnTick()
  {
   static int tick_counter=0;
//---
   tick_counter++;
   Comment("\nBefore unloading expert advisor ",__FILE__," left",
```

© 2000-2025, MetaQuotes Ltd.

```
            (ticks_to_close-tick_counter)," ticks");
//--- 以前
   if(tick_counter>=ticks_to_close)
     {
      ExpertRemove();
      Print(TimeCurrent(),": ",__FUNCTION__," expert advisor will be unloaded");
     }
   Print("tick_counter =",tick_counter);
//---
  }
//+------------------------------------------------------------------+
```

另见

程序运行，客户端事件

© 2000-2025, MetaQuotes Ltd.

# GetPointer

函数返回 指针 对象。

```
void*  GetPointer(
   any_class anyobject      // 任何类的对象
   );
```

参量

*anyobject*

[in] 任一类的对象。

返回值

函数返回对象指针

注释

只有分类对象有指标，例如架构和简单类型变量没有指标。分类对象不能使用new ()操作建立，但是，例如，数组对象会自动建立，直到产生指标。但是该指标会是自动类型 POINTER_AUTOMATIC，因此delete()操作不能应用，该类型指标与动态 POINTER_DYNAMIC 类型指标不同。

结构类型变量和简单类型没有指标，也禁止应用GetPointer()函数。也禁止以函数论据传递指标，在此情况下，编译器会通知出现一个错误。

想要调用引起程序 危险终止 的错误指标，这就是使用指标之前调用 CheckPointer() 函数的原因，如下情况指标错误：

- 指标等于 NULL；
- 使用 delete 操作对象删除；

该函数可用来有效检验指标，保证非零值，该指标可用于访问。

示例：

```
//+------------------------------------------------------------------+
//|                                              Check_GetPointer.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+------------------------------------------------------------------+
//| 执行列表元素的类                                                 |
//+------------------------------------------------------------------+
class CItem
  {
   int               m_id;
   string            m_comment;
```

© 2000-2025, MetaQuotes Ltd.

```
   CItem*            m_next;
public:
                     CItem() { m_id=0; m_comment=NULL; m_next=NULL; }
                    ~CItem() { Print("Destructor of ",m_id,
                                     (CheckPointer(GetPointer(this))==POINTER_DYNAMIC)
                                     "dynamic":"non-dynamic"); }
   void              Initialize(int id,string comm) { m_id=id; m_comment=comm; }
   void              PrintMe() { Print(__FUNCTION__,":",m_id,m_comment); }
   int               Identifier() { return(m_id); }
   CItem*            Next() {return(m_next); }
   void              Next(CItem *item) { m_next=item; }
  };
//+------------------------------------------------------------------+
//| 列表中最简单的类                                                 |
//+------------------------------------------------------------------+
class CMyList
  {
   CItem*            m_items;
public:
                     CMyList() { m_items=NULL; }
                    ~CMyList() { Destroy(); }
    bool             InsertToBegin(CItem* item);
    void             Destroy();
  };
//+------------------------------------------------------------------+
//| 从开始嵌入列表元素                                               |
//+------------------------------------------------------------------+
bool CMyList::InsertToBegin(CItem* item)
  {
   if(CheckPointer(item)==POINTER_INVALID) return(false);
//---
   item.Next(m_items);
   m_items=item;
//---
   return(true);
  }
//+------------------------------------------------------------------+
//| 通过删除元素删除列表                                             |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- 循环工作服务指针
   CItem* item;
//--- 循环检测并删除动态指针
   while(CheckPointer(m_items)!=POINTER_INVALID)
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          Print("Dynamyc object ",item.Identifier()," to be deleted");
          delete (item);
         }
       else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
      }
//---
  }
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   CMyList list;
   CItem   items[10];
   CItem*  item;
//--- 创建并加入一个动态对象指针到列表
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(100,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- 添加自动指针到列表
   for(int i=0; i<10; i++)
     {
      items[i].Initialize(i,"automatic");
      items[i].PrintMe();
      item=GetPointer(items[i]);
      if(CheckPointer(item)!=POINTER_INVALID)
         list.InsertToBegin(item);
     }
//--- 在列表开始添加一个以上动态对象指针
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(200,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- 删除所有列表元素
   list.Destroy();
//--- 脚本结束后所有列表元素都被删除
//--- 末端看到专家标签
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

对象指针，检测对象指针，对象删除操作符的删除

© 2000-2025, MetaQuotes Ltd.

# GetTickCount

自从程序开始，GetTickCount()函数返回已过去的毫秒的数量。

```
uint  GetTickCount();
```

返回值

无符号整型值

注释

系统定时器限制计数器，时间以一种无符号的整数来存储，因此，如果电脑不间断的工作，每49.7就会溢满。

例如：

```
#define MAX_SIZE 40
//+------------------------------------------------------------------+
//| 测量40个裴波那契数列计算时间的脚本                                |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 牢记初始值
   uint start=GetTickCount();
//--- 在裴波那契数列获得下一个数列的变量
   long fib=0;
//--- 循环计算来自裴波那契数列的指定数列量
   for(int i=0;i<MAX_SIZE;i++) fib=TestFibo(i);
//--- 获取毫秒为单位的所用时间
   uint time=GetTickCount()-start;
//--- 导出信息到专家日志
   PrintFormat("Calculating %d first Fibonacci numbers took %d ms",MAX_SIZE,time);
//--- 已完成脚本
   return;
  }
//+------------------------------------------------------------------+
//| 通过数列号获得裴波那契数列的函数                                  |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- 裴波那契数列的第一个成员
   if(n<2) return(1);
//--- 所有其他成员通过以下公式计算
   return(TestFibo(n-2)+TestFibo(n-1));
  }
```

See also

日期和时间, EventSetMillisecondTimer, GetTickCount64, GetMicrosecondCount

© 2000-2025, MetaQuotes Ltd.

# GetTickCount64

GetTickCount64()函数返回系统启动后经过的毫秒数。

```
ulong  GetTickCount64();
```

返回值

一个ulong类型值。

注意

计数器仅限于系统计时器的精度，通常以10-16毫秒精度返回结果。与GetTickCount不同，它是uint类型，其内容在计算机连续运行的情况下，每49.7天超限一次，而GetTickCount64()可用于无限的计算机操作时间且不会超限。

示例：

```
#define MAX_SIZE 40

//+------------------------------------------------------------------+
//| 用于测量计算40个Fibo数的时间的脚本                                |
//+------------------------------------------------------------------+
void OnStart()
  {
   long fib_array[MAX_SIZE];

//--- 存储初始值
   ulong start=GetTickCount64();
//--- 一个循环，在这个循环中，我们从Fibo序列中计算出给定数量的数字
   for(int i=0;i<MAX_SIZE;i++)
      fib_array[i]=TestFibo(i);
//--- 以毫秒为单位获取花费的时间
   ulong time=GetTickCount64()-start;

//--- 在EA日志中显示错误信息
   ArrayPrint(fib_array);
   PrintFormat("Calculating the first %d Fibonacci numbers took %I64u ms",MAX_SIZE,tim
  }
//+------------------------------------------------------------------+
//| 通过序列号获取Fibo号码的函数                                      |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- 斐波序列的第一个成员
   if(n<2)
      return(1);
//--- 所有后续成员都使用该方程式计算
   return(TestFibo(n-2)+TestFibo(n-1));
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另见

日期和时间、EventSetMillisecondTimer、GetTickCount、GetMicrosecondCount

© 2000-2025, MetaQuotes Ltd.

# GetMicrosecondCount

GetMicrosecondCount() 函数返回MQL5程序开始后过去的微秒数。

```
ulong  GetMicrosecondCount();
```

返回值

无符号长整型的值。

示例：

```
//+------------------------------------------------------------------+
//| 测试函数                                                         |
//+------------------------------------------------------------------+
void Test()
  {
   int    res_int=0;
   double res_double=0;
//---
   for(int i=0;i<10000;i++)
     {
      res_int+=i*i;
      res_int++;
      res_double+=i*i;
      res_double++;
     }
  }
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   uint   ui=0,ui_max=0,ui_min=INT_MAX;
   ulong  ul=0,ul_max=0,ul_min=INT_MAX;
//--- 测量次数
   for(int count=0;count<1000;count++)
     {
      uint  ui_res=0;
      ulong ul_res=0;
      //---
      for(int n=0;n<2;n++)
        {
         //--- 选择测量类型
         if(n==0) ui=GetTickCount();
         else     ul=GetMicrosecondCount();
         //--- 执行代码
         Test();
         //--- 添加测量结果（根据类型）
         if(n==0) ui_res+=GetTickCount()-ui;
```

© 2000-2025, MetaQuotes Ltd.

```
         else     ul_res+=GetMicrosecondCount()-ul;
        }
      //--- 为两个测量计算最小和最大的时间值
      if(ui_min>ui_res) ui_min=ui_res;
      if(ui_max<ui_res) ui_max=ui_res;
      if(ul_min>ul_res) ul_min=ul_res;
      if(ul_max<ul_res) ul_max=ul_res;
     }
//---
   Print("GetTickCount error(msec): ",ui_max-ui_min);
   Print("GetMicrosecondCount error(msec): ",DoubleToString((ul_max-ul_min)/1000.0,2))
  }
```

另见

日期和时间, GetTickCount, GetTickCount64

© 2000-2025, MetaQuotes Ltd.

# MessageBox

建立并显示对话框并管理它，一个对话框包括信息和页眉，任意预定义标签和命令按钮的组合。

```
int  MessageBox(
   string  text,              // 信息文本
   string  caption=NULL,      // 对话框表头
   int     flags=0            // 对话框按键设置
   );
```

参量

*text*

[in] 文本，包括信息输出。

*caption=NULL*

[in] 可选文本出现在方框中，如果参量是空的，EA交易名称在方框中显示。

*flags=0*

[in] 可选项 标记 规定外观和信息框的行为，标签可与特殊标签组结合。

返回值

如果函数成功执行，返回值是 MessageBox() 返回代码中值的一个。

注释

该函数不可用于自定义指标，因为MessageBox()调用在等待用户响应时，会一直暂停执行线程。由于每个交易品种的所有指标都在一个线程中执行，因此暂停会造成无法操作该交易品种的所有时间周期的所有图表。

MessageBox()函数在策略测试中不工作。

示例：

```
//+------------------------------------------------------------------+
//| EA显示包含进一步工作请求的MessageBox窗口                         |
//| 当一系列未盈利交易达到指定数量时                                 |
//| 等待指定的柱形图数量并显示MessageBox                             |
//| 请求继续工作。                                                   |
//| 为了进行检查，我们需要手动打开和关闭多个持仓                     |
//| 亏损，因为EA无法控制                                             |
//| 它自己的“持仓”通过幻数)。                                        |
//+------------------------------------------------------------------+

//--- 输入参数
input uint InpMaxLossDeals       =  3;    // 最大亏损交易
input uint InpInactivityNumBars =  5;    // 不活动的柱形图数

//--- 全局变量
```

© 2000-2025, MetaQuotes Ltd.

```
bool     ExtFirstStart=true;              // 第一启动标识
bool     ExtFlag=true;                    // 允许EA工作的标识
uint     ExtNumLoss;                      // 连续未盈利交易数
datetime ExtTimeLastLoss;                 // 最后一笔亏损平仓的时间

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 获得连续亏损交易的数量和最后一笔亏损平仓的时间
   ExtNumLoss=GetNumLosingTradesInRow(ExtTimeLastLoss);

   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   Comment("");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 确定自该系列中最后一个收盘亏损持仓以来已经过多少柱形图
   int bars_remaining=iBarShift(Symbol(),PERIOD_CURRENT,ExtTimeLastLoss);

//--- 如果这是第一次启动
   if(ExtFirstStart)
     {
      //--- 如果在一系列未盈利持仓之后已经通过指定数量的柱形图，则设置EA操作标识
      if(bars_remaining>(int)InpInactivityNumBars)
         ExtFlag=true;
      ExtFirstStart=false;
     }

//--- 如果EA操作标识被禁用
   if(!ExtFlag)
     {
      Comment(StringFormat("The advisor is stopped for %d bars. Num Loss positions: %u
                           (InpInactivityNumBars-bars_remaining),ExtNumLoss,TimeToStrin
      //--- 如果在一系列未盈利持仓之后已经通过指定数量的柱形图
      if(bars_remaining>(int)InpInactivityNumBars)
        {
         //--- 显示带有指定文本和窗口标题的MessageBox窗口
         //--- 请求窗口有两个 "是(Yes)"/ "否(No)"按键和一个带问号的图标。
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 默认选择 "是(Yes)"按键。
         string mb_text="The specified number of bars of EA inactivity have passed.\n
         string mb_caption="Please note";
         int    mb_id=MessageBox(mb_text,mb_caption,MB_YESNO|MB_ICONQUESTION|MB_DEFBUT
         //--- 如果来自MessageBox的返回代码是按下的 "Yes"按键，则设置EA操作标识
         if(mb_id==IDYES)
           {
            ExtFlag=true;
            return;
           }
        }
      //--- EA操作标识被禁用，退出OnTick()
      return;
     }

//---  设置EA操作标识 - EA按照下面代码提供的方式工作
   Comment(StringFormat("The advisor is working. Num Loss positions: %u, Time last los
           ExtNumLoss,TimeToString(ExtTimeLastLoss,TIME_DATE|TIME_MINUTES|TIME_SECONDS
  }
//+------------------------------------------------------------------+
//| TradeTransaction函数                                             |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
  {
//--- 如果交易事务类型是向历史记录添加交易
   if(trans.type==TRADE_TRANSACTION_DEAL_ADD)
     {
      //--- 获得一个交易工单，并按工单从列表中选择一笔交易
      ulong deal_ticket=trans.deal;
      if(HistoryDealSelect(deal_ticket))
        {
         //--- 如果这是一个市场退出交易，得到连续亏损交易的数量和最后一次交易的时间
         ENUM_DEAL_ENTRY entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(deal_ticket,DEAI
         if(entry==DEAL_ENTRY_OUT || entry==DEAL_ENTRY_INOUT || entry==DEAL_ENTRY_OUT_
            ExtNumLoss=GetNumLosingTradesInRow(ExtTimeLastLoss);
        }
     }

//--- 如果连续亏损的交易数量大于指定的值，则启用EA操作标识
   if(ExtNumLoss>=InpMaxLossDeals && ExtFlag)
     {
      //--- 显示包含指定文本和窗口标题的MessageBox窗口
      //--- 请求窗口有两个 "是(Yes)"/ "否(No)"按键和一个带感叹号的图标。
      //--- 默认情况下选择 "否(No)"按键。
      string mb_text="The number of losing trades has reached the specified maximum. T
      string mb_caption="Attention!";
      int    mb_id=MessageBox(mb_text,mb_caption,MB_YESNO|MB_ICONQUESTION|MB_DEFBUTTON
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果来自MessageBox的返回代码是按下的 "No"按键，，则禁用EA操作标识
      if(mb_id==IDNO)
         ExtFlag=false;
     }
  }
//+------------------------------------------------------------------+
//| 返回连续未盈利交易的数量                                         |
//| 以及最后一笔亏损平仓的时间                                       |
//+------------------------------------------------------------------+
uint GetNumLosingTradesInRow(datetime &time_last_deal)
  {
//--- 选择整个历史
   if(!HistorySelect(0,TimeCurrent()))
      return(0);

//--- 通过循环中的历史交易列表获得下一个交易工单
   uint res=0;
   uint total=HistoryDealsTotal();
   for(int i=(int)total-1; i>=0; i--)
     {
      ulong deal_ticket=HistoryDealGetTicket(i);
      if(deal_ticket>0)
        {
         //--- 如果交易不是为了退出持仓，那么继续下一笔交易
         ENUM_DEAL_ENTRY entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(deal_ticket,DEAI
         if(entry!=DEAL_ENTRY_OUT && entry!=DEAL_ENTRY_OUT_BY && entry!=DEAL_ENTRY_INC
            continue;
         //--- 如果平仓的结果是获利，则中断循环
         if(!IsClosePositionWithLoss(deal_ticket))
            break;
         //--- 增加连续亏损交易的计数器
         res++;
         //--- 将最大交易时间写入变量 查找最后一个）
         datetime deal_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
         if(deal_time>time_last_deal)
            time_last_deal=deal_time;
        }
     }

//--- 返回连续亏损的数量
   return(res);
  }
//+------------------------------------------------------------------+
//| 返回亏损平仓标识                                                 |
//+------------------------------------------------------------------+
bool IsClosePositionWithLoss(const ulong deal_ticket)
  {
//--- 获取影响交易利润的属性值
   double profit=HistoryDealGetDouble(deal_ticket,DEAL_PROFIT);
```

© 2000-2025, MetaQuotes Ltd.

```
   double comission=HistoryDealGetDouble(deal_ticket,DEAL_COMMISSION);
   double swap=HistoryDealGetDouble(deal_ticket,DEAL_SWAP);
   double fee=HistoryDealGetDouble(deal_ticket,DEAL_FEE);

//--- 返回标识，表示接收到的属性总价值为负值
   return(profit+comission+swap+fee<0);
  }
```

© 2000-2025, MetaQuotes Ltd.

# PeriodSeconds

在一个周期内函数二次返回数字。

```
int  PeriodSeconds(
   ENUM_TIMEFRAMES  period=PERIOD_CURRENT      // 图表周期
   );
```

参量

*period=PERIOD_CURRENT*

[in] ENUM_TIMEFRAMES中的图表周期值，如果参量未标明，在程序运行中，返回当前图表周期二次数字。

返回值

所选周期中的二次数字。

示例：

```
//--- 输入参数
input ENUM_TIMEFRAMES InpPeriod1 = PERIOD_CURRENT;    // First Period
input ENUM_TIMEFRAMES InpPeriod2 = PERIOD_M1;         // Second Period

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取InpPeriod1和InpPeriod2图表周期中的秒数
   int sec1=PeriodSeconds(InpPeriod1);
   int sec2=PeriodSeconds(InpPeriod2);
//--- 在日志中显示接收到的值
   PrintFormat("Seconds in period %s: %lu, in period %s: %lu",TimeframeDescription(Inp
//--- 计算InpPeriod1图表周期的柱形图中包含了多少个InpPeriod2图表周期的柱形图
   int res=sec1/sec2;
   if(res==0)
      res=1;
//--- 在日志中显示获取到的值
   PrintFormat("One bar %s contains %d bars %s",TimeframeDescription(InpPeriod1),res,T
   /*
  结果:
   Seconds in period M5: 300, in period M1: 60
   One bar M5 contains 5 bars M1
   */
  }
//+------------------------------------------------------------------+
//| 返回时间周期名称                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
string TimeframeDescription(const ENUM_TIMEFRAMES period)
  {
   return(StringSubstr(EnumToString(period==PERIOD_CURRENT ? Period() : period), 7));
  }
```

另见

_Period, 图表时间表, 日期和时间, 对象可见性

© 2000-2025, MetaQuotes Ltd.

# PlaySound

用来播放声音文件。

```
bool  PlaySound(
   string  filename      // 文件名
   );
```

参量

*filename*

[in] 到声音文件的路径。如果 filenam e=NULL，停止回放。

返回值

true -如果发现文件，否则 - false.

注释

文件加载到term inal_directory\Sounds 的子目录中，只能播放wav类型文件。

通过NULL参数调用PlaySound() 停止回放。

PlaySound() 函数在策略测试中不工作。

示例：

```
#include <Trade\Trade.mqh>
#define MAGIC (123)

//--- 输入参数
input string           InpFileNameOK  = "ok.wav";               // 成功音频文件
input string           InpFileNameErr = "timeout.wav";          // 错误音频文件
input ENUM_ORDER_TYPE  InpOrderType   = ORDER_TYPE_BUY_LIMIT;   // 订单类型
input double           InpLots        = 0.1;                    // 手数

//--- 全局变量
CTrade ExtTrade;

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 根据交易品种设置来设置幻数和订单执行类型
   ExtTrade.SetExpertMagicNumber(MAGIC);
   ExtTrade.SetTypeFillingBySymbol(Symbol());
//--- 调用带有声音回放的下订单或开仓函数
   OrderSendWithAudio();
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 该函数下订单或开仓                                                |
//| 以及播放成功或失败的声音                                          |
//+------------------------------------------------------------------+
void OrderSendWithAudio(void)
  {
   bool    res=true;
   MqlTick tick= {};

   ResetLastError();
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      PlaySound(InpFileNameErr);
      return;
     }
//--- 向服务器发送请求
   switch(InpOrderType)
     {
      case ORDER_TYPE_BUY :
         res=ExtTrade.Buy(InpLots);
         break;
      case ORDER_TYPE_BUY_LIMIT :
         res=ExtTrade.BuyLimit(InpLots,NormalizeDouble(tick.ask-100*Point(),Digits()))
         break;
      case ORDER_TYPE_BUY_STOP :
         res=ExtTrade.BuyStop(InpLots,NormalizeDouble(tick.ask+100*Point(),Digits()));
         break;
      case ORDER_TYPE_SELL :
         res=ExtTrade.Sell(InpLots);
         break;
      case ORDER_TYPE_SELL_LIMIT :
         res=ExtTrade.SellLimit(InpLots,NormalizeDouble(tick.bid+100*Point(),Digits())
         break;
      case ORDER_TYPE_SELL_STOP :
         res=ExtTrade.SellStop(InpLots,NormalizeDouble(tick.bid-100*Point(),Digits()))
         break;
      default :
         res=false;
     }
   if(!res)
      Print("Error ",GetLastError());
   Print(ExtTrade.ResultRetcodeDescription());

//--- 如果请求被接受，则播放ok.wav声音
   if(ExtTrade.ResultRetcode()==TRADE_RETCODE_DONE)
      PlaySound(InpFileNameOK);
   else
      PlaySound(InpFileNameErr);
  }
```

© 2000-2025, MetaQuotes Ltd.

另见

资源

© 2000-2025, MetaQuotes Ltd.

# Print

在EA交易日志中输入消息。参量可以是各种类型。

```
void  Print(
   argument,      // 第一值
   ...            // 下一值
   );
```

参量

...

[in] 逗号分割任意值，参量数不能超过64

注释

数组不能通过Print()函数传递，数组必须按元素一个接一个输入

双精度型数据多余小数点后16位字节显示，或以传统或以科学角度输出，根据接入口的紧凑性。浮点类型数据在小数点后5位字节输出，输出的实型数据和另一精确度的预先格式，使用 PrintFormat() 函数。

布尔型数据的输出以true或者false标准，数据以 YYYY.MM.DD HH:MI:SS为标准，在另一版本表示数据，使用 TimeToString()，如果颜色类型显示在操作中，颜色类型数据以RGB，或颜色名称来返回。

Print() 函数在优化期间在策略测试中不工作。

示例：

```
void OnStart()
  {
//--- 用Print()导出DBL_MAX，这相当于PrintFormat(%%.16G,DBL_MAX)
   Print("---- how DBL_MAX looks like -----");
   Print("Print(DBL_MAX)=",DBL_MAX);
//--- 现在用PrintFormat()导出DBL_MAX number
   PrintFormat("PrintFormat(%%.16G,DBL_MAX)=%.16G",DBL_MAX);
//--- 导出到专家日志
// Print(DBL_MAX)=1.797693134862316e+308
// PrintFormat(%.16G,DBL_MAX)=1.797693134862316E+308

//--- 请看浮动如何导出
   float c=(float)M_PI; // 我们应该明确目标类型
   Print("c=",c, "    Pi=",M_PI, "    (float)M_PI=",(float)M_PI);
// c=3.14159    Pi=3.141592653589793    (float)M_PI=3.14159

//--- 显示真实类型算术运算会发生什么
   double a=7,b=200;
   Print("---- Before arithmetic operations");
   Print("a=",a,"   b=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- a 除以 b (7/200)
   a=a/b;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 现在模仿恢复b变量的值
   b=7.0/a; // 期待 b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 - 但却不同
//--- 导出最新计算的b值
   Print("----- After arithmetic operations");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- 导出到专家日志
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.9999999999999716 (可见b不再等于200.0)

//--- 创建一个非常小的值 epsilon=1E-013
   double epsilon=1e-13;
   Print("---- Create a very small value");
   Print("epsilon=",epsilon); // 获得 epsilon=1E-013
//--- 现在从b减去epsilon 并且导出值到专家日志
   b=b-epsilon;
//--- 使用两种方式
   Print("---- After subtracting epsilon from the b variable");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- 导出到专家日志
// Print(b)=199.9999999999999  (现在减去epsilon以后的b值不能四舍五入到200)
// Print(DoubleToString(b,16))=199.9999999999998578
//    (现在减去epsilon以后的b值不能四舍五入到200)
  }
```

另 见

DoubleToString, StringFormat

© 2000-2025, MetaQuotes Ltd.

# PrintFormat

交易品种的形式和进入，EA交易的值与当前形式保持一致。

```
void  PrintFormat(
   string format_string,    // 格式行
   ...                      // 简单类型值
   );
```

参量

*format_string*

[in] A格式包括简单交易品种，如果模式线接下来是参数，也包括模式规格

*...*

[in] 简单类型的任何值都被逗号分开，参数的全部数量都不能超过64个，包括格式化路线

返回值

字符串。

注释

PrintFormat() 函数在优化期间在策略测试中不工作。

数字，订单和参数类型必须明确匹配后置修饰语的设置，否则打印结果无法确定。替代 PrintFormat() 您可以使用 printf()。

如果格式路线随后是参量，这条路线一定包括这些参量表示输出的格式，规格总是以百分比符号(%)开始

格式路线从左至右阅读，当遇见第一个格式 (如果有的话)，在格式线路后的第一个参量值根据当前规格改变并输出。第二个格式根据第二个参量改变并输出，以此类推直到格式化线路结束。

格式规格如下形式：

**%[flags][width][.precision][{h | l | ll | I32 | I64}]type**

每个格式规格的域或者是一个简单的交易对象，或者表示一般选项的数字，最简单的规格只包括百分号(%)和定义成 输出参量类型 的交易品种 (例如%s)，如果在格式线上需要输出当前符号，使用格式规格%%.

## 标记

| 标记 | 描述 | 默认动作 |
| --- | --- | --- |
| - (减号) | 活字宽度左对齐 | 右对齐 |
| + (加号) | + or的输出-符号类型的符号值 | 符号只在负值中显示 |
| 0 (零) | 在当前 宽度 输出值前添加0，如果0标签指定整数格式 (i, u, x, X, o, d)，设立精确度格式 (例如， | 什么都不添加 |

© 2000-2025, MetaQuotes Ltd.

| 标记 | 描述 | 默认动作 |
| --- | --- | --- |
| | %04.d)，然后忽略0 | |
| 空间 | 如果有符号和正值，在输出值中显示空间 | 不插入空间 |
| # | 如果连用格式 **o, x** 或者 **X**,在输出0值之前，分别添加0x 或者 0X | 什么都不添加 |
| | 如果连用格式 **e, E, a** 或者 **A**,值会显示小数点 | 如果小数部分非零，显示小数点 |
| | 如果连用格式**g**或者**G**，标签在输出值中存在小数点，阻止切断前导零，当一起使用格式**c, d, i, u, s**时，忽视标签# | 如果小数部分非零，显示小数点，切断前导零 |

## 宽度

建立非负小数，格式化值中的交易品种输出的最小数量。如果输出交易品种数量少于活字宽度，类似的空间数量依据队列 (标签-)从左至右添加，如果有标签0 (0)，类似0数值在输出值之前添加，如果输出值数量比活字宽度大，输出值不会被切断

如果星号(*)规定宽度，整型值应该在传递参量列表中类似位置显示，用来说明输出值的宽度。

## 精确度

非负的小数建立在输出精确度-小数点后几位数字。与规定宽度不同，精确度规格切断部分类型。

使用规格精确度与不同的板式 类型 不同

| 类型 | 描述 | 默认动作 |
| --- | --- | --- |
| **a, A** | 精确度说明在小数点后建立的数字数量 | 默认精确度-6 |
| **c, C** | 不使用 | |
| **d, i, u, o, x, X** | 集合输出数字的最少数量，如果数字数量与常量差不多但小于精确度，在输出值左边加上0，如果输出数字的数量比规定精确度多，输出值不能中断。 | 默认精确度-1 |
| **e, E, f** | 在小数点后设置若干输出数量，最后的数字四舍五入 | 默认精确度-6，如果建立的精确度是0或者小数部分空缺，不显示小数点 |
| **g, G** | 设立有意义数字的最大数量 | 输出六位有意义数字 |
| **s** | 建立线型输出交易品种数量，如果线的长度超过精确度，线就会切断 | 整个线路输出 |

© 2000-2025, MetaQuotes Ltd.

```
PrintFormat("1. %s", _Symbol);
PrintFormat("2. %.3s", _Symbol);
int length=4;
PrintFormat("3. %.*s", length, _Symbol);
/*
1. EURUSD
2. EUR
3. EURU
/
```

## h | l | ll | I32 | I64

数据大小说明，以参量传递。

| 参量类型 | 使用前缀 | 类型共同说明符 |
|---|---|---|
| int | l(小写字母 L) | d, i, o, x, or X |
| uint | l(小写字母 L) | o, u, x, or X |
| long | ll(两个小写字母 L) | d, i, o, x, or X |
| short | h | d, i, o, x, or X |
| ushort | h | o, u, x, or X |
| int | I32 | d, i, o, x, or X |
| uint | I32 | o, u, x, or X |
| long | I64 | d, i, o, x, or X |
| ulong | I64 | o, u, x, or X |

## 类型

类型说明符只在强制域格式化输出

| 符号 | 类型 | 输出格式 |
|---|---|---|
| c | int | 短型交易品种（统一码） |
| C | int | 图表型交易品种(ANSI) |
| d | int | 小数整数记号 |
| i | int | 小数整数记号 |
| o | int | 未标签的八进制整数 |
| u | int | 未标签的小数整数 |

© 2000-2025, MetaQuotes Ltd.

| 符号 | 类型 | 输出格式 |
| --- | --- | --- |
| x | int | 未标签的十六进制整数，使用"abcdef" |
| X | int | 未标签的十六进制整数，使用 "ABCDEF" |
| e | double | 在 [-]d.ddde[sign]ddd, where d-十进制数字，dddd 一位或更多小数位，ddd -三位数字觉得类型大小 格式中真值，标志-正负号 |
| E | double | 与e格式相同，除了典型标志通过大写字母标出 €代替e） |
| f | double | 格式中的真值[-] dddd.dddd,dddd 一位或更多小数位数，在小数点前显示数字的数量取决于符号值的大小，小数点后的数字数量取决于需求 |
| g | double | f或者e格式的输出真值取决于输出量更紧密 |
| G | double | F或者E的输出真值取决于输出量更紧密 |
| a | double | 格式中的实数 [-]0xh.hhhh p±dd,h.hhhh 十六进制格式的尾数，使用"abcdef",dd 一个或者更多典型数字，小数位的数量由 精确度说明 决定。 |
| A | double | 格式中的实数[-]0xh.hhhh P±dd, h.hhhh-十六进制格式的尾数，使用"ABSDEF",dd 一个或者更多典型数字，小数位的数量由 精确度说明 决定。 |
| s | string | 路线输出 |

取代PrintFormat()，可以使用printf()。

示例：

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
//--- 交易服务器名称
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- 账号
   int login=(int)AccountInfoInteger(ACCOUNT_LOGIN);
//--- 长整型值输出
   long leverage=AccountInfoInteger(ACCOUNT_LEVERAGE);
   PrintFormat("%s %d: leverage = 1:%I64d",
               server,login,leverage);
//--- 账户货币
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- 小数点后两位的双精度值输出
   double equity=AccountInfoDouble(ACCOUNT_EQUITY);
   PrintFormat("%s %d: account equity = %.2f %s",
               server,login,equity,currency);
//--- +/-符号强制输出的双精度值输出
   double profit=AccountInfoDouble(ACCOUNT_PROFIT);
   PrintFormat("%s %d: current result for open positions = %+.2f %s",
               server,login,profit,currency);
//--- 小数点后可变数位的双精度值输出
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   string format_string=StringFormat("%%s: point value  = %%.%df",_Digits);
   PrintFormat(format_string,_Symbol,point_value);
//--- 整型值输出
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   PrintFormat("%s: current spread in points = %d ",
               _Symbol,spread);
//--- 小数点后17个有意义数位的科学（浮点）格式双精度值输出
   PrintFormat("DBL_MAX = %.17e",DBL_MAX);
//--- 小数点后17个有意义数位的科学（浮点）格式双精度值输出
   PrintFormat("EMPTY_VALUE = %.17e",EMPTY_VALUE);
//--- 以默认精度使用PrintFormat()输出
   PrintFormat("PrintFormat(EMPTY_VALUE) = %e",EMPTY_VALUE);
//--- 使用Print()简单输出
   Print("Print(EMPTY_VALUE) = ",EMPTY_VALUE);
/* 执行结果
   MetaQuotes-Demo 1889998: leverage = 1:100
   MetaQuotes-Demo 1889998: account equity = 22139.86 USD
   MetaQuotes-Demo 1889998: current result for open positions = +174.00 USD
   EURUSD: point value  = 0.00001
   EURUSD: current spread in points = 12
   DBL_MAX = 1.79769313486231570e+308
   EMPTY_VALUE = 1.79769313486231570e+308
   PrintFormat(EMPTY_VALUE) = 1.797693e+308
   Print(EMPTY_VALUE) = 1.797693134862316e+308
*/
  }
```

另见

StringFormat，DoubleToString，真实类型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# ResetLastError

设立预先定义常量 _LastError 的值是0。

```
void  ResetLastError();
```

返回值

没有返回值

注释

如果需要注释函数 GetLastError() 是zero _LastError 变量，在函数调用之前 错误 外观检测后通常调用ResetLastError()。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在调用函数之前重置最后一个错误代码，
//--- 否则，GetLastError()可能返回之前的错误代码
   long lres=SymbolInfoInteger("123456",SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
   lres=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
   ResetLastError();
   lres=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# ResourceCreate

基于数据设置创建图像资源。有两种函数变量：
根据文件创建资源

```
bool  ResourceCreate(
   const string      resource_name,       // 资源名称
   const string      path                 // 文件的相对路径
   );
```

基于像素数组创建资源

```
bool  ResourceCreate(
   const string      resource_name,       // 资源名称
   const uint&       data[],              // 数据设置为数组
   uint              img_width,           // 图像资源的宽度
   uint              img_height,          // 图像资源的高度
   uint              data_xoffset,        // 水平向右偏移图像左上角
   uint              data_yoffset,        // 垂直向下偏移图像左上角
   uint              data_width,          // 基于数据设置的图像总宽度
   ENUM_COLOR_FORMAT color_format         // 颜色处理方案
   );
```

参数

*resource_name*
[in] 资源名称。

*data[][]*
[in] 用于创建完整图像的一维或二维数组。

*img_width*
[in] 矩形图像区域的像素宽度会以图像的形式置于资源中。它不能大于*data_width*的值。

*img_height*
[in] 矩形图像区域的像素高度会以图像的形式置于资源中。

*data_xoffset*
[in] 图像矩形区域水平向右偏移。

*data_yoffset*
[in] 图像矩形区域垂直向右偏移。

*data_width*
[in] 只需一维数组。它表示全宽幅的数据设置图像。如果*data_width*=0，假设它等于*img_width*。对于二维数组，参数被忽略并假设等于二维*data[]*数组。

*color_format*
[in] 颜色处理方案，来自ENUM_COLOR_FORMAT枚举的值。

返回值

© 2000-2025, MetaQuotes Ltd.

如果成功返回true，否则false。若要获得有关错误的信息，请调用GetLastError() 函数。可能会发生以下错误：

- 4015 -ERR_RESOURCE_NAME_DUPLICATED (动态和静态资源的相同名称)
- 4016 -ERR_RESOURCE_NOT_FOUND (资源没找到)
- 4017 -ERR_RESOURCE_UNSUPPORTED_TYPE (这种类型的资源不支持)
- 4018 -ERR_RESOURCE_NAME_IS_TOO_LONG (资源名称太长)

**ENUM_COLOR_FORMAT**

| 标识符 | 描述 |
|---|---|
| COLOR_FORMAT_XRGB_NOALPHA | 阿尔法通道的组件被忽略 |
| COLOR_FORMAT_ARGB_RAW | 程序端不处理颜色组件 (用户必须正确设置) |
| COLOR_FORMAT_ARGB_NORMALIZE | 程序端不处理颜色组件 |

示例：

```
//+------------------------------------------------------------------+
//| 更新图形资源数据                                                 |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- 如果传递的维度为零，则离开
   if(width==0 || height==0)
      return;
//--- 更新资源数据并重新绘制图表
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALIZ
      ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明图形资源的参数
   string rc_name="Resource";
   uint   rc_width=100;
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- 设置像素数组的大小
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
//--- 用透明颜色填充像素数组，并在此基础上创建图形资源
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- 查看已创建的图形资源。
//--- 获取当前柱形图的时间和价格数据
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- 使用最后报价和时间的坐标创建位图对象
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 将创建的位图对象的宽度和高度设置为图形资源的宽度和高度。
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- 将前面为位图对象创建的图形资源指定为图像文件
//--- 在这种情况下，为了指示所使用的图形资源的名称，我们需要在其名称之前添加“::”
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- 设置DodgerBlue的透明度为200
   uint clr=ColorToARGB(clrDodgerBlue,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data,clr);
//--- 更新图形资源数据
   Update(rc_name,rc_data,rc_width,rc_height,true);
  }
```

另见

资源, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

© 2000-2025, MetaQuotes Ltd.

# ResourceFree

该函数删除动态创建的资源 (释放分配的内存)。

```
bool  ResourceFree(
   const string  resource_name      // 资源名称
   );
```

参数

*resource_name*

[in] 资源 名称应以"::"开头。

返回值

如果成功返回true，否则false。若要获得有关错误的信息，请调用GetLastError() 函数。

注意

当积极使用资源时，ResourceFree() 允许mql5应用开发者们管理内存消耗。从内存删除的绑定资源的图形对象 会在删除后正确显示。然而，新创建的图形对象 (OBJ_BITMAP 和 OBJ_BITMAP_LABEL) 将不能使用已删除的资源。

函数仅删除通过程序创建的动态资源。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明图形资源的参数
   string rc_name="Resource";
   uint   rc_width=100;
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- 设置像素数组的大小
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- 用透明颜色填充像素数组，并在此基础上创建图形资源
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- 查看已创建的图形资源。
//--- 获取当前柱形图的时间和价格数据
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- 使用最后报价和时间的坐标创建位图对象
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 将创建的位图对象的宽度和高度设置为图形资源的宽度和高度。
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- 将前面为位图对象创建的图形资源指定为图像文件
//--- 在这种情况下，为了指示所使用的图形资源的名称，我们需要在其名称之前添加“::”
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- 设置DodgerBlue的透明度为200
   uint clr=ColorToARGB(clrDodgerBlue,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data,clr);
//--- 更新图形资源数据
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- 等待3秒，改变图像颜色
   Print("Wait 3 seconds before changing color");
   Sleep(3000);
//--- 设置OrangeRed的透明度为200
   Print("Change color");
   clr=ColorToARGB(clrOrangeRed,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data,clr);
//--- 更新图形资源数据
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- 等待3秒，释放图形资源
   Print("Wait 3 seconds before ResourceFree()");
   Sleep(3000);
```

© 2000-2025, MetaQuotes Ltd.

```
   bool res=ResourceFree("::"+rc_name);
   Print("ResourceFree: ",res);

//--- 在释放资源后尝试改变颜色
   Print("Trying to change color to GreenYellow after ResourceFree()");
//--- 设置GreenYellow的透明度为200
   clr=ColorToARGB(clrGreenYellow,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data,clr);
//--- 更新图形资源数据 图像保留，但不能更改颜色）
   Update(rc_name,rc_data,rc_width,rc_height,true);
   Print("The color has not changed because the resource has been released");

//--- 等待3秒，删除位图对象
   Print("Wait 3 seconds before deleting the Bitmap object");
   Sleep(3000);
   Print("Delete Bitmap object");
   ObjectDelete(0,obj_name);
  }
//+------------------------------------------------------------------+
//| 更新图形资源数据                                                 |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- 如果传递的维度为零，则离开
   if(width==0 || height==0)
      return;
//--- 更新资源数据并重新绘制图表
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
```

另见

资源, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

© 2000-2025, MetaQuotes Ltd.

# ResourceReadImage

函数阅读来自通过ResourceCreate()函数创建的 或 编译期间保存在 EX5 文件的图形资源的数据。

```
bool  ResourceReadImage(
   const string      resource_name,        // 用于阅读的图形资源名称
   uint&             data[],               // 用于从资源接收数据的数组
   uint&             width,                // 用于在资源中接收图像宽度
   uint&             height,               // 用于在资源中接收图像高度
   );
```

参数

*resource_name*

[in] 包含图像的图形资源的名称。若要访问它的资源，以简单形式使用名称"::resourcename"。如果我们从编译的EX5文件下载资源，全名应该和相对于MQL5目录的路径，文件和资源名称一起使用 -"path\\filename.ex5::resourcename"。

*data[][]*

[in] 接收图形资源数据的一维或二维数组。

*img_width*

[out] 图形资源图像的像素宽度。

*img_height*

[out] 图形资源图像的像素高度。

返回值

如果成功返回true，否则false。若要获得有关错误的信息，请调用GetLastError() 函数。

注意

如果*data[]* 数组被用于创建图形资源，COLOR_FORMAT_ARGB_NORMALIZE 或 COLOR_FORMAT_XRGB_NOALPHA 颜色格式应该被使用。

如果*data[]* 数组是二维数组并且它的第二维度小于X (宽度)图形资源大小，那么ResourceReadImage() 函数返回false 并且阅读不被执行。但是如果资源存在，实际图像大小返回到宽度和高度参数。这将允许采取另外尝试接收资源数据。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明图形资源的参数
   string rc_name="Resource";
   uint   rc_width=100;
```

© 2000-2025, MetaQuotes Ltd.

```
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- 设置像素数组的大小
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- 用透明颜色填充像素数组，并在此基础上创建图形资源
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- 查看已创建的图形资源。
//--- 获取当前柱形图的时间和价格数据
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- 使用最后报价和时间的坐标创建位图对象
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 将创建的位图对象的宽度和高度设置为图形资源的宽度和高度。
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- 将前面为位图对象创建的图形资源指定为图像文件
//--- 在这种情况下，为了指示所使用的图形资源的名称，我们需要在其名称之前添加“::”
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- 设置DodgerBlue的透明度为200
   uint clr=ColorToARGB(clrDodgerBlue,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data,clr);
//--- 更新图形资源数据
```

© 2000-2025, MetaQuotes Ltd.

```
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- 等待3秒再读取图形资源数据
   Print("Wait 3 seconds before ResourceReadImage()");
   Sleep(3000);
//--- 将图像从资源读入新的像素数组
   uint rc_data_copy[];
   uint w=0,h=0;
   ResetLastError();
   if(!ResourceReadImage("::"+rc_name,rc_data_copy,w,h))
     {
      Print("ResourceReadImage() failed. Error code: ",GetLastError());
      return;
     }

//--- 设置OrangeRed的透明度为200
   clr=ColorToARGB(clrOrangeRed,200);
//--- 用设置的颜色填充图形资源的整个像素数组，并在此基础上创建新的图形资源
   ArrayInitialize(rc_data_copy,clr);
   if(!ResourceCreate(rc_name+"Copy",rc_data_copy,rc_width,rc_height,0,0,0,COLOR_FORMA
     {
      Print("New ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created new recource array: ",rc_data_copy.Size());

//--- 使用最后报价和时间的坐标创建 "图形标签"对象
   string obj_name2="BitmapLabel";
   if(!ObjectCreate(0,obj_name2,OBJ_BITMAP_LABEL,0,0,0))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 使用之前接收到的价格和时间获取屏幕坐标
   int x=0,y=0;
   if(!ChartTimePriceToXY(0,0,tick.time,tick.bid,x,y))
     {
      Print("New ChartTimePriceToXY() failed. Error code: ",GetLastError());
      return;
     }
//--- 将创建的图形标签对象的宽度和高度设置为图形资源的宽度和高度。
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name2,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name2,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name2,OBJPROP_ANCHOR,ANCHOR_LEFT_UPPER);
   ObjectSetInteger(0,obj_name2,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(0,obj_name2,OBJPROP_YDISTANCE,y);
//--- 将复制的图形资源设置为图形标签对象的图像文件
//--- 在这种情况下，为了指示所使用的图形资源的名称，我们需要在其名称之前添加 ":"
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetString(0,obj_name2,OBJPROP_BMPFILE,"::"+rc_name+"Copy");

//--- 更改新图形标签对象的颜色
   Print("Wait 3 seconds before changing color to GreenYellow");
   Sleep(3000);
//--- 设置GreenYellow的透明度为200
   clr=ColorToARGB(clrGreenYellow,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(rc_data_copy,clr);
//--- 更新图形资源数据
   Update(rc_name+"Copy",rc_data_copy,rc_width,rc_height,true);

//--- 等待3秒，删除资源和两个对象
   Print("Wait 3 seconds before deleting both objects");
   Sleep(3000);
   Print("Deleting Resource and all Bitmap objects");
   ResourceFree("::"+rc_name);
   ResourceFree("::"+rc_name+"Copy");
   ObjectDelete(0,obj_name);
   ObjectDelete(0,obj_name2);
  }
//+------------------------------------------------------------------+
//| 更新图形资源数据                                                 |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- 如果传递的维度为零，则离开
   if(width==0 || height==0)
      return;
//--- 更新资源数据并重新绘制图表
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
```

另见

资源, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

© 2000-2025, MetaQuotes Ltd.

# ResourceSave

保存资源到指定文件。

```
bool  ResourceSave(
   const string  resource_name      // 资源名称
   const string  file_name          // 文件名称
   );
```

参数

*resource_name*

[in] 资源名称，必须以"::"开始。

*file_name*

[in] 相对于MQL5\Files文件名称。

返回值

true -如果成功，否则false。对于错误信息，调用GetLastError()。

注意

函数始终覆盖文件并且必要情况下，以文件名称创建所有所需的中间目录。

示例：

```
//--- 图形资源参数
string ExtResName="Resource";
int    ExtResWidth=100;
int    ExtResHeight=100;
uint   ExtResData[];
int    ExtResSize=ExtResWidth*ExtResHeight;

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   ResetLastError();
//--- 设置像素数组的大小
   if(ArrayResize(ExtResData,ExtResSize)!=ExtResSize)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- 用透明颜色填充像素数组，并在此基础上创建图形资源
   ArrayInitialize(ExtResData,0x00FFFFFF);
   if(!ResourceCreate(ExtResName,ExtResData,ExtResWidth,ExtResHeight,0,0,0,COLOR_FORMA
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",ExtResData.Size());

//--- 查看已创建的图形资源。
//--- 获取当前柱形图的时间和价格数据
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- 使用最后报价和时间的坐标创建位图对象
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 将创建的位图对象的宽度和高度设置为图形资源的宽度和高度。
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,ExtResWidth);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,ExtResHeight);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- 将前面为位图对象创建的图形资源指定为图像文件
//--- 在这种情况下，为了指示所使用的图形资源的名称，我们需要在其名称之前添加“::”
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+ExtResName);

//--- 设置GreenYellow的透明度为200
   uint clr=ColorToARGB(clrGreenYellow,200);
//--- 用设置的颜色填充图形资源的整个像素数组
   ArrayInitialize(ExtResData,clr);
//--- 使用DodgerBlue颜色绘制网格
   for(int x=0;x<ExtResWidth;x+=9)
      LineVertical(x,0,ExtResHeight,ColorToARGB(clrDodgerBlue,200));
   for(int y=0;y<ExtResHeight;y+=9)
      LineHorizontal(0,ExtResWidth,y,ColorToARGB(clrDodgerBlue,200));
//--- 更新图形资源数据
   Update(ExtResName,ExtResData,ExtResWidth,ExtResHeight,true);

//--- 等待3秒后，再删除资源和图形对象
   Print("Wait 3 seconds before deleting the resource and graphic object");
   Sleep(3000);

//--- 将资源保存到文件
   ResetLastError();
   if(!ResourceSave("::"+ExtResName,"ResourceSave\\"+ExtResName+".bmp"))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("ResourceSave() failed. Error code: ",GetLastError());
      return;
     }
//--- 删除资源和图形对象图像
   if(!ResourceFree("::"+ExtResName))
     {
      Print("ResourceFree() failed. Error code: ",GetLastError());
      return;
     }
   if(!ObjectDelete(0,obj_name))
     {
      Print("ObjectDelete() failed. Error code: ",GetLastError());
      return;
     }

//--- 现在使用向左移动图像宽度的最后报价和时间坐标创建 "图形标签"对象
//--- 使用之前接收到的价格和时间获取屏幕坐标
   int x=0,y=0;
   if(!ChartTimePriceToXY(0,0,tick.time,tick.bid,x,y))
     {
      Print("ChartTimePriceToXY() failed. Error code: ",GetLastError());
      return;
     }
   obj_name="BitmapLabel";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP_LABEL,0,0,0))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- 将对象定位点设置为其中心。
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- 将最后报价和时间的坐标设置为图形标签对象，并向左移动图像宽度
   ObjectSetInteger(0,obj_name,OBJPROP_XDISTANCE,x-ExtResWidth);
   ObjectSetInteger(0,obj_name,OBJPROP_YDISTANCE,y);
//--- 将以前保存为BMP图像的资源指定为图形标签对象的图像文件
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"\\Files\\ResourceSave\\"+ExtResName+".b
//--- 更新图表
   ChartRedraw();

//--- 等待3秒后，再删除图形标签对象
   Print("Wait 3 seconds before deleting the new graphic object");
   Sleep(3000);
   if(!ObjectDelete(0,obj_name))
      Print("ObjectDelete() failed. Error code: ",GetLastError());
  }
//+------------------------------------------------------------------+
//| 更新图形资源数据                                                 |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 如果传递的维度为零，则离开
   if(width==0 || height==0)
      return;
//--- 更新资源数据并重新绘制图表
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 绘制垂直线                                                       |
//+------------------------------------------------------------------+
void LineVertical(int x,int y1,int y2,const uint clr)
  {
   int tmp;
//--- 按Y排序
   if(y1>y2)
     {
      tmp=y1;
      y1 =y2;
      y2 =tmp;
     }
//--- 图像边界外的线
   if(y2<0 || y1>=ExtResHeight || x<0 || x>=ExtResWidth)
      return;
//--- 保持在图像边界内
   if(y1<0)
      y1=0;
   if(y2>=ExtResHeight)
      y2=ExtResHeight-1;
//--- 绘制线
   int index=y1*ExtResWidth+x;
   for(int i=y1; i<=y2; i++,index+=ExtResWidth)
      ExtResData[index]=clr;
  }
//+------------------------------------------------------------------+
//| 绘制水平线                                                       |
//+------------------------------------------------------------------+
void LineHorizontal(int x1,int x2,int y,const uint clr)
  {
   int tmp;
//--- 按X排序
   if(x1>x2)
     {
      tmp=x1;
      x1 =x2;
      x2 =tmp;
     }
//--- 图像边界外的线
   if(x2<0 || x1>=ExtResWidth || y<0 || y>=ExtResHeight)
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
//--- 保持在图像边界内
   if(x1<0)
      x1=0;
   if(x2>=ExtResWidth)
      x2=ExtResWidth-1;
//--- 绘制线
   ArrayFill(ExtResData,y*ExtResWidth+x1,(x2-x1)+1,clr);
  }
```

另见

资源, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

© 2000-2025, MetaQuotes Ltd.

# SetReturnError

设置完成操作时返回程序端进程的代码。

```
void  SetReturnError(
   int ret_code      // 客户端完成代码
   );
```

参数

*ret_code*

[in] 完成操作时由客户端进程返回的代码。

返回值

无返回值。

注意

使用SetReturnError()函数设置指定的 *ret_code*返回代码有利于分析通过命令行启用程序端时程序操作完成的原因。

与TerminalClose()不同，SetReturnError()函数未完成程序端操作。相反，它只设置完成后返回程序端进程的代码。

如果SetReturnError()函数被调用多次和/或从不同的MQL5程序调用，程序端返回最后一次设置的返回代码。

除下列情况外，程序端进程完成后返回设置代码：

- 执行期间发生严重错误；
- 已调用以指定代码发行程序端操作完成命令的TerminalClose(int ret_code)函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   matrix matrix_a =
     {
        {-3.474589, 1.106384, -9.091977,-3.925227 },
        {-5.522139, 2.366887,-15.162351,-6.357512 },
        { 8.394926,-2.960067, 22.292115, 9.524129 },
        { 7.803242,-2.080287, 19.217706, 8.186645 }
     };
   matrix matrix_l(4,4);
   matrix matrix_u(4,4);

//--- LU分解
```

© 2000-2025, MetaQuotes Ltd.

```
   matrix_a.LU(matrix_l,matrix_u);

//--- 检查是否A = L * U
   matrix matrix_lu=matrix_l.MatMul(matrix_u);
   int    compare_errors=(int)matrix_a.Compare(matrix_lu,1e-29);
   Print("MatrixCompare errors=",compare_errors);

//--- 完成后，客户端将返回比较两个矩阵时的错误数
   SetReturnError(compare_errors);
  }
```

另见

程序运行、运行错误、未初始化原因代码、TerminalClose

© 2000-2025, MetaQuotes Ltd.

# SetUserError

创建预先参量 _LastError，使其值等于 ERR_USER_ERROR_FIRST + user_error

```
void  SetUserError(
   ushort user_error,   // 错误号
   );
```

参量

*user_error*

[in] 用户设置的错误编号。

返回值

没有返回值

注释

在使用SetUserError(user_error) 函数发生错误后，GetLastError() 返回值等于 ERR_USER_ERROR_FIRST + user_error.

注释：

```
void OnStart()
  {
//--- 设置错误号 65537=(ERR_USER_ERROR_FIRST +1)
   SetUserError(1);
//--- 获得上一个错误代码
   Print("GetLastError = ",GetLastError());
/*
   Result
   GetLastError = 65537
*/
  }
```

© 2000-2025, MetaQuotes Ltd.

# Sleep

在当前执行的EA交易或脚本中，该函数是指在指定的时间间隔内暂停交易业务。

```
void  Sleep(
   int  milliseconds      // 间隔
   );
```

参量

*milliseconds*

[in] 毫秒之内的内部延期

返回值

没有返回值

注释

The Sleep()函数不能被自定义指标调用，因为指标是以界面线路通过并不能够放慢速度，智能交易每0.1 second秒会检测。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在图表评论区显示从10到1的倒计时
   for(int i=10; i>0 && !IsStopped(); i--)
     {
      Comment(StringFormat("Wait %u seconds",i));
      Sleep(1000);
     }
//--- 在 "到达"评论中编辑一段文字，描述脚本的目的
   string text="This was a test showing how the Sleep() function works";
   string mess="";
   for(int i=0; i<(int)text.Length(); i++)
     {
      mess+=ShortToString(text.GetChar(i));
      Sleep(100);
      Comment(mess);
     }
//--- 再见...
   Sleep(1000);
   for(int i=0; i<6; i++)
     {
      mess=(i % 2 == 0 ? "" : "  Bye!");
      Comment(mess);
```

© 2000-2025, MetaQuotes Ltd.

```
      Sleep(300);
     }
//--- 删除图表上的文本
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# TerminalClose

该函数命令服务器完整操作。

```
bool  TerminalClose(
   int ret_code      // 关闭客户端代码
   );
```

参量

*ret_code*

[in] 返回代码，在操作完成后通过客户端返回。

返回值

函数成功true，否则-false。

注释

TerminalClose() 函数并不能立即阻止客户端，它只命令客户端完成操作。

EA交易的代码叫做TerminalClose()，它来执行立即操作（比如，所有先前开放文件在正常模式下都要先关闭），该函数的调用通过 返回操作符执行。

*ret_code* 参数程序分析原因返回的必要代码，在命令提示符启动后，客户端操作终止

示例：

```
//--- 输入参量
input int  tiks_before=500; // 订单号数目直到终止
input int  pips_to_go=15;   //  pips 间的距离
input int  seconds_st=50;   // 给予EA交易的秒数
//--- 全局
datetime   launch_time;
int        tick_counter=0;
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(__FUNCTION__," reason code = ",reason);
   Comment("");
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
   static double first_bid=0.0;
   MqlTick       tick;
```

© 2000-2025, MetaQuotes Ltd.

```
   double         distance;
//---
   SymbolInfoTick(_Symbol,tick);
   tick_counter++;
   if(first_bid==0.0)
     {
      launch_time=tick.time;
      first_bid=tick.bid;
      Print("first_bid =",first_bid);
      return;
     }
//---  pips 的价格距离
   distance=(tick.bid-first_bid)/_Point;
//--- 通过EA操作显示告示
   string comm="From the moment of start:\r\n\x25CF elapsed seconds: "+
               IntegerToString(tick.time-launch_time)+" ;"+
               "\r\n\x25CF ticks received: "+(string)tick_counter+" ;"+
               "\r\n\x25CF price went in points: "+StringFormat("%G",distance);
   Comment(comm);
//--- 检测关闭程序端状态的部分
   if(tick_counter>=tiks_before)
      TerminalClose(0);     // 推出订单号计数器
   if(distance>pips_to_go)
      TerminalClose(1);     // 同于pips_to_go增长pips数量
   if(distance<-pips_to_go)
      TerminalClose(-1);    // 同于pips_to_go下降pips的数量
   if(tick.time-launch_time>seconds_st)
      TerminalClose(100);  // 超时终止
//---
  }
```

另见

程序运行，执行错误，无法初始化原因

© 2000-2025, MetaQuotes Ltd.

# TesterHideIndicators

设置显示/隐藏EA中使用的指标的模式。这个函数仅用于在测试期间管理已用指标的可见性。

```
void  TesterHideIndicators(
   bool      hide     // 标识
   );
```

参数

*隐藏*

[in]用于测试时隐藏指标的标识。设置隐藏已创建指标为true，否则为false。

返回值

无。

注意

默认情况下，已测试EA创建的所有指标都显示在可视测试图表上。此外，这些指标还显示在测试完成后自动打开的图表上。TesterHideIndicators()函数允许开发人员实现禁止显示已用指标的功能。

若要禁止显示测试EA时已用的指标，请在创建EA句柄之前调用TesterHideIndicators()等于true - 在此后创建的所有指标都被标记为隐藏标识。这些指标在可视测试期间和测试完成后自动打开的图表上都不显示。

若要禁用新创建指标的隐藏模式，请调用TesterHideIndicators() 等于false。只有从测试EA直接生成的指标才可以显示在测试图表上。这个规则只适用于<data_folder>MQL5\Profiles\Templates没有单一模板的情况下。

如果<data_folder>MQL5\Profiles\Templates文件目录包含一个特殊模板<EA_name>.tpl，那么只有该模板的指标可以在可视测试期间和测试图表上显示。在这种情况下，测试EA中无已用指标可显示。即使在EA代码中调用TesterHideIndicators()等于true，这种情况下仍然存在。

如果<data_folder>MQL5\Profiles\Templates目录不包含特殊模板<EA_name>.tpl而是以tester.tpl来替代，那么没有被TesterHideIndicators()函数禁用的来自tester.tpl的指标和来自EA的指标会在可视测试期间和测试图表上显示。如果没有tester.tpl模板，则使用default.tpl模板的指标来替代。

如果策略测试发现了不合适的模板(<EA_name>.tpl, tester.tpl或default.tpl)，那么EA中使用的指标的显示则完全由TesterHideIndicators() 函数管理。

例如：

```
bool CSampleExpert::InitIndicators(void)
  {
   TesterHideIndicators(true);
//--- 创建MACD指标
   if(m_handle_macd==INVALID_HANDLE)
      if((m_handle_macd=iMACD(NULL,0,12,26,9,PRICE_CLOSE))==INVALID_HANDLE)
        {
         printf("Error creating MACD indicator");
         return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
         }
   TesterHideIndicators(false);
//--- 创建EMA指标并将其添加到集
   if(m_handle_ema==INVALID_HANDLE)
      if((m_handle_ema=iMA(NULL,0,InpMATrendPeriod,0,MODE_EMA,PRICE_CLOSE))==INVALID_H
        {
         printf("Error creating EMA indicator");
         return(false);
        }
//--- 成功
   return(true);
  }
```

另见

IndicatorRelease

© 2000-2025, MetaQuotes Ltd.

# TesterStatistics

函数返回基于测试结果计算的指定统计参数的值。

```
double  TesterStatistics(
   ENUM_STATISTICS statistic_id      // ID
   );
```

参数

*statistic_id*

[in] 来自 ENUM_STATISTICS 枚举的统计参数的ID。

返回值

来自测试结果的统计参数的值。

注意

在测试器中函数可以在OnTester() 或 OnDeinit() 内部调用。其他情况下结果不确定。

示例：

```
// 基于标准"MACD Sample.mq5"文件的EA
// 在Tester事件处理程序中显示TesterStatistics()的结果
#define MACD_MAGIC 1234502
//---
#include <Trade\Trade.mqh>
#include <Trade\SymbolInfo.mqh>
#include <Trade\PositionInfo.mqh>
#include <Trade\AccountInfo.mqh>
//---
input double InpLots          =0.1; // 手数
input int    InpTakeProfit    =50;  // 盈利 点数)
input int    InpTrailingStop  =30;  // 追踪止损 点数)
input int    InpMACDOpenLevel =3;   // MACD开盘价 点数)
input int    InpMACDCloseLevel=2;   // MACD收盘价 点数)
input int    InpMATrendPeriod =26;  // MA趋势线周期
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit(void)
  {
//--- 创建所有必要的对象
   if(!ExtExpert.Init())
      return(INIT_FAILED);
//--- 成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| Expert new tick handling function                                |
//+------------------------------------------------------------------+
void OnTick(void)
  {
   static datetime limit_time=0; // 考虑'timeout'的最后一次调用时间
//--- 如果时间超过指定的limit_time值
   if(TimeCurrent()>=limit_time)
     {
      //--- 检查数据
      if(Bars(Symbol(),Period())>2*InpMATrendPeriod)
        {
         //--- 如果成功,将limit_time增加'timeout'秒
         if(ExtExpert.Processing())
            limit_time=TimeCurrent()+ExtTimeOut;
        }
     }
  }
//+------------------------------------------------------------------+
//| Expert tester handling function                                  |
//+------------------------------------------------------------------+
double OnTester(void)
  {
   double ret=TesterStatistics(STAT_PROFIT_FACTOR);
   double profit=TesterStatistics(STAT_PROFIT);
   int    trades_total=(int)TesterStatistics(STAT_TRADES);
   int    profit_total=(int)TesterStatistics(STAT_PROFIT_TRADES);
   int    loss_total=(int)TesterStatistics(STAT_LOSS_TRADES);
   PrintFormat("%s: Profit = %.2f, trades total: %lu, profit trades total: %lu, loss t
   return(ret);
   /*
  结果:
   OnTester: Profit = 209.84, trades total: 13, profit trades total: 11, loss trades t
   final balance 10209.84 USD
   OnTester result 3.020606644198363
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TesterStop

当测试时，给予程序操作完成命令。

```
void  TesterStop();
```

返回值

无返回值。

注意

TesterStop()函数是为测试代理上的EA例行早期关闭而设计的 -例如，当达到指定数量的亏损交易或预先设定的下降水平时。

TesterStop()调用被认为是测试的正常完成，因此OnTester() 函数被调用，将全部累积的交易统计数据和优化准则值提交到策略测试。

在策略测试中调用 ExpertRemove()还意味着正常测试完成并允许获取交易统计数据，但EA从代理内存中卸载。在这种情况下，对下一组参数执行传递需要一段时间来重新加载重新。因此，TesterStop() 是早期例行完成测试的首选选项。

示例：

```
//--- 定义
#define BALANCE_LOSS_STOP  100.0        // 结余回撤值，在该值停止测试
#define EQUITY_LOSS_STOP   100.0        // 净值回撤值，在该值停止测试

//--- 输入参数
input  double  InpLots         =  0.1;  // 手数
input  uint    InpStopLoss     =  50;   // 止损点数
input  uint    InpTakeProfit   =  150;  // 止盈点数
sinput ulong   InpMagic        =  123;  // 幻数
sinput ulong   InpDeviation    =  5;    // 偏差
//--- 全局变量
CTrade      trade;                      // 交易类实例
CSymbolInfo symb;                       // 交易品种类实例
CAccountInfo account;                   // 交易账户类实例
...
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- 成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 更新当前报价
   if(!symb.RefreshRates())
      return;
   ...

//--- 如果余额或净值下降幅度超过BALANCE_LOSS_STOP和EQUITY_LOSS_STOP宏替换中指定的值，
//--- 则测试被认为不成功，并且调用TesterStop()函数
//--- 检查结余损失是否超过BALANCE_LOSS_STOP
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()<balance_prev-BALANCE_LOSS_STOP)
        {
         PrintFormat("The initial balance of %.2f %s decreased by %.2f %s, and now has
         TesterStop();
         /*
        结果：
         The initial balance of 10000.00 USD decreased by 100.10 USD, and now has a va
         TesterStop() called on 9% of testing interval
         */
        }
     }
//--- 检查净值损失是否超过EQUITY_LOSS_STOP
   if(equity_prev!=account.Equity())
     {
      if(account.Equity()<equity_prev-EQUITY_LOSS_STOP)
        {
         PrintFormat("The initial equity of %.2f %s decreased by %.2f %s, and now has
         TesterStop();
         /*
        结果：
         The initial equity of 10000.00 USD decreased by 100.10 USD, and now has a val
         TesterStop() called on 9% of testing interval
         */
        }
     }
  }
```

另见

程序运行、测试交易策略、ExpertRemove、SetReturnError

© 2000-2025, MetaQuotes Ltd.

# TesterDeposit

在测试期间模拟存款的特殊函数。它在一些资金管理系统中使用。

```
bool  TesterDeposit(
   double money      // 存款金额
   );
```

参数

*money*

[in] 以入金货币存入账户的资金。

返回值

如果成功返回true，否则返回false。

示例：

```
//--- 定义
#define BALANCE_LOSS_DEPOSIT  100.0    // 结余回撤值，在该值时资金将被存入测试器中的账户

//--- 输入参数
input  double  InpLots         =  0.1;  // 手数
input  uint    InpStopLoss     =  50;   // 止损点数
input  uint    InpTakeProfit   =  150;  // 止盈点数
sinput ulong   InpMagic        =  123;  // 幻数
sinput ulong   InpDeviation    =  5;    // 偏差
//--- 全局变量
CTrade      trade;                      // 交易类实例
CSymbolInfo symb;                       // 交易品种类实例
CAccountInfo account;                   // 交易账户类实例
...
double      balance_dep_summ;           //结余充值的总金额
uint        balance_dep_total;          // 结余充值次数
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- 保存初始结余值
   balance_prev=account.Balance();
   balance_dep_summ=0;
   balance_dep_total=0;
//--- 成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 更新当前报价
   if(!symb.RefreshRates())
      return;
   ...

//--- 如果结余下降幅度超过BALANCE_LOSS_DEPOSIT宏替换中指定的值，
//--- 则需要充值账户并调用TesterDeposit()函数
//--- 检查结余损失是否超过BALANCE_LOSS_DEPOSIT
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()<balance_prev-BALANCE_LOSS_DEPOSIT)
        {
         double loss=balance_prev-account.Balance();
         PrintFormat("The initial balance of %.2f %s decreased by %.2f %s. It is neces
         if(TesterDeposit(loss))
           {
            balance_dep_total++;
            balance_dep_summ+=loss;
            balance_prev=account.Balance();
            PrintFormat("Funds have been deposited into the account. Account balance:
            PrintFormat("Total deposits: %lu. Amount of deposits: %.2f %s.",balance_de
           }
         /*
        结果：
         The initial balance of 10000.00 USD decreased by 116.00 USD. It is necessary
         deal #45 balance 116.00 [deposit] done
         Funds have been deposited into the account. Account balance: 10000.00 USD.
         Total deposits: 1. Amount of deposits: 116.00 USD.
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| 测试函数                                                         |
//+------------------------------------------------------------------+
double OnTester()
  {
//--- 将以货币形式表示的最大结余回撤设置为输出处理程序值
   double ret=TesterStatistics(STAT_BALANCE_DD);
//--- 在日志中显示有关回撤、存款次数及其总金额的消息
   PrintFormat("%s: Maximum balance drawdown in money: %.2f %s. Total deposits: %lu. A
//--- 返回结果
   return(ret);
   /*
  结果：
   OnTester: Maximum balance drawdown in money: 5188.50 USD. Total deposits: 46. Amour
```

© 2000-2025, MetaQuotes Ltd.

```
   final balance 4867.50 USD
   OnTester result 5188.5
   */
  }
```

另见

TesterWithdrawal

© 2000-2025, MetaQuotes Ltd.

# TesterWithdrawal

在测试过程中特殊函数模仿取消内存操作，可以用于资产管理系统中。

```
bool  TesterWithdrawal(
   double money      // 出金总额
   );
```

参量

*money*

[in] 要求出金的总钱数（用如今货币）

返回值

成功，返回 true，否则 - false.

示例：

```
//--- 定义
#define BALANCE_PROFIT_WITHDRAWAL   5  // 结余盈利值，在该值时资金从测试器的账户中提取

//--- 输入参数
input  double  InpLots         =  0.1;  // 手数
input  uint    InpStopLoss     =  50;   // 止损点数
input  uint    InpTakeProfit   =  150;  // 止盈点数
sinput ulong   InpMagic        =  123;  // 幻数
sinput ulong   InpDeviation    =  5;    // 偏差
//--- 全局变量
CTrade      trade;                      // 交易类实例
CSymbolInfo symb;                       // 交易品种类实例
CAccountInfo account;                   // 交易账户类实例
...
double      balance_op_sum;             // 结余操作的总额
uint        balance_op_total;           // 结余操作次数
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- 保存初始结余值
   balance_prev=account.Balance();
   balance_op_sum=0;
   balance_op_total=0;
//--- 成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 更新当前报价
   if(!symb.RefreshRates())
      return;
   ...

//--- 如果结余盈利超过在BALANCE_PROFIT_WITHDRAWAL宏替换中指定的当前结余值，
//--- 则需要从账户中提取这些资金。调用TesterWithdrawal()函数。
//--- 检查结余盈利是否超过BALANCE_PROFIT_WITHDRAWAL
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()>balance_prev+BALANCE_PROFIT_WITHDRAWAL)
        {
         double profit=account.Balance()-balance_prev;
         PrintFormat("The account balance has been increased by %.2f %s. Need to witho
         if(TesterWithdrawal(profit))
           {
            balance_op_total++;
            balance_op_summ+=profit;
            balance_prev=account.Balance();
            PrintFormat("Funds have been withdrawn from the account. Account balance:
            PrintFormat("Total withdrawals: %lu. Amount of withdrawals: %.2f %s.",bala
           }
         /*
        结果：
         The account balance has been increased by 21.00 USD. Need to withdraw these f
         deal #13 balance -21.00 [withdrawal] done
         Funds have been withdrawn from the account. Account balance: 10000.00 USD.
         Total withdrawals: 1. Amount of withdrawals: 21.00 USD.
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| 测试函数                                                         |
//+------------------------------------------------------------------+
double OnTester()
  {
//--- 将以货币形式表示的最大结余回撤设置为输出处理程序值
   double ret=TesterStatistics(STAT_BALANCE_DD);
//--- 在日志中显示有关回撤、提款次数及其总金额的消息
   PrintFormat("%s: Maximum balance drawdown in money: %.2f %s. Total withdrawals: %lu
//--- 返回结果
   return(ret);
   /*
  结果：
   OnTester: Maximum balance drawdown in money: 5188.50 USD. Total withdrawals: 2. Amo
```

© 2000-2025, MetaQuotes Ltd.

```
   final balance 4867.50 USD
   OnTester result 5188.5
   */
  }
```

另见

TesterDeposit

© 2000-2025, MetaQuotes Ltd.

# TranslateKey

通过考虑到当前输入语言和控制键状态的虚拟键码返回Unicode字符。

```
short  TranslateKey(
   int  key_code      // 接收Unicode字符的键码
   );
```

参数

*key_code*

[in] 键码。

返回值

成功转换情况下的Unicode字符。错误情况下，函数返回 -1。

注意

函数使用ToUnicodeEx将用户按下的键码转换为Unicode字符。如果没有触发ToUnicodeEx则可能发生错误 例如，试图接收SHIFT键字符时。

例如：

```
void OnChartEvent(const int id,const long& lparam,const double& dparam,const string& s
  {
   if(id==CHARTEVENT_KEYDOWN)
     {
      short sym=TranslateKey((int)lparam);
      //--- 如果输入的字符成功转换为Unicode
      if(sym>0)
         Print(sym,"'",ShortToString(sym),"'");
      else
         Print("Error in TranslateKey for key=",lparam);
     }
  }
```

另见

客户端事件, OnChartEvent

© 2000-2025, MetaQuotes Ltd.

# ZeroMemory

通过引用，函数重置变量

```
void  ZeroMemory(
   void & variable      // 变量重置
   );
```

参量

*变量*

[in][out] 变量通过引用传递，然后重置（从0开始初始化）

返回值

没有返回值

注释

如果函数参数是字符串，调用为NULL值
对于简单类型或数组，和由这些类型组成的结构和分类，都要简单重置
该对象包括串型和动态数组，ZeroMemory() 调用每个元素

对于没有被参量修饰语保护的任何数组，所有元素都归零。

复杂对象的数字，ZeroMemory() 调用每个元素

ZeroMemory() 不能应用到受 成员 或 继承算法 保护的分类中

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化字符串
   string str="Test ZeroMemory func";
//--- 在对该行应用ZeroMemory()之前，将该行发送到日志
   PrintFormat("The line before applying ZeroMemory() to it: '%s'",str);
//--- 重置字符串并将结果发送到日志
   ZeroMemory(str);
   Print("The same line after applying ZeroMemory() to it: '",str,"'");
  /*
  结果：
   The line before applying ZeroMemory() to it: 'Test ZeroMemory func'
   The same line after applying ZeroMemory() to it: ''
  */

//--- 声明并初始化int类型的变量
   int var=123;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在对该行应用ZeroMemory()之前，将该行发送到日志
   PrintFormat("\nThe integer variable before applying ZeroMemory() to it: %d",var);
//--- 重置变量并将结果发送到日志
   ZeroMemory(var);
   PrintFormat("The same variable after applying ZeroMemory() to it: %d",var);
  /*
  结果：
   The integer variable before applying ZeroMemory() to it: 123
   The same variable after applying ZeroMemory() to it: 0
  */

//--- 声明并初始化int类型数组
   int arr[]={0,1,2,3,4,5,6,7,8,9};
//--- 在对数组应用ZeroMemory()之前将其发送到日志
   Print("\nThe integer array before applying ZeroMemory() to it:");
   ArrayPrint(arr);
//--- 重置数组并将结果发送到日志
   ZeroMemory(arr);
   Print("The same array after applying ZeroMemory() to it:");
   ArrayPrint(arr);
  /*
  结果：
   The integer array before applying ZeroMemory() to it:
   0 1 2 3 4 5 6 7 8 9
   The same array after applying ZeroMemory() to it:
   0 0 0 0 0 0 0 0 0 0
  */

//--- 声明两个字段的结构 - 字符串字段和整数字段
   struct STest
     {
      string   var_string;
      long     var_long;
     };
//--- 声明并初始化STest结构类型的数组
   STest arr_struct[]={ {"0",0}, {"1",1}, {"2",2}, {"3",3} };
//--- 在对数组应用ZeroMemory()之前将其发送到日志
   Print("\nThe array struct before applying ZeroMemory() to it:");
   ArrayPrint(arr_struct);
//--- 重置结构数组并将结果发送到日志
   ZeroMemory(arr_struct);
   Print("The same array struct after applying ZeroMemory() to it:");
   ArrayPrint(arr_struct);
  /*
  结果：
   The array struct before applying ZeroMemory() to it:
       [var_string] [var_long]
   [0] "0"                   0
   [1] "1"                   1
```

© 2000-2025, MetaQuotes Ltd.

```
   [2] "2"                   2
   [3] "3"                   3
   The same array struct after applying ZeroMemory() to it:
       [var_string] [var_long]
   [0] null                  0
   [1] null                  0
   [2] null                  0
   [3] null                  0
  */
  }
```

© 2000-2025, MetaQuotes Ltd.

# 使用数组的函数组

数组的最大维数为四维。每个维数被索引编为从0 至*dimension_size*-1。 事实上，第一维数组的50 个，在调用时第一个数组显示为[0],最后一个数组显示为[49]。

| 函数 | 功能 |
| --- | --- |
| ArrayBsearch | 在第一函数维度索引第一组发现的元素 |
| ArrayCopy | 复制一组数组到另一组 |
| ArrayFree | 解放任意动态数组的缓冲区并建立0维度大小 |
| ArrayCompare | 返回比较简单类型或无复杂对象的自定义结构的两种数组的结果 |
| ArrayGetAsSeries | 检测数组标引的导向 |
| ArrayInitialize | 给数字数组所有元素一个单独的值 |
| ArrayFill | 以指定值填充数组 |
| ArrayIsSeries | 检测是否数组是时序列 |
| ArrayIsDynamic | 检测是否数组是动态的 |
| ArrayMaximum | 检测元素的最大值 |
| ArrayPrint | 将简单类型或简单结构的数组打印到日志 |
| ArrayMinimum | 检测元素的最小值 |
| ArrayRange | 在数组的指定维度返回元素数量 |
| ArrayResize | 在数组的第一维尺寸设定新大小 |
| ArrayInsert | 从指定索引开始，将来自源数组的指定元素数量插入到接收数组 |
| ArrayRemove | 从指定索引开始移除数组中指定的元素数量 |
| ArrayReverse | 从指定索引开始反转数组中指定的元素数量 |
| ArraySetAsSeries | 指明数组标引的导向 |
| ArraySize | 返回数组中元素的数量 |
| ArraySort | 通过第一维度排列数值数组 |
| ArraySwap | 交换相同类型两个动态数组的内容 |
| ArrayToFP16 | 将float或double类型数组复制到给定格式的ushort类型的数组中 |
| ArrayToFP8 | 将float或double类型数组复制到给定格式的uchar类型的数组中 |
| ArrayFromFP16 | 将ushort类型数组复制到给定格式的float或double类型的数组中 |
| ArrayFromFP8 | 将uchar类型数组复制到给定格式的float或double类型的数组中 |

© 2000-2025, MetaQuotes Ltd.

# ArrayBsearch

在升序排列的多维数值数组中搜索指定的值。搜索通过第一维度元素执行。

搜寻双精度数组

```
int  ArrayBsearch(
   const double&    array[],   // 用来搜索的数组
   double           value      // 搜索什么
   );
```

搜寻浮点型数组

```
int  ArrayBsearch(
   const float&    array[],   // 用来搜索的数组
   float           value      // 搜索什么
   );
```

搜寻长整型数组

```
int  ArrayBsearch(
   const long&    array[],   // 用来搜索的数组
   long           value      // 搜索什么
   );
```

搜寻整型数组

```
int  ArrayBsearch(
   const int&    array[],   // 用来搜索的数组
   int           value      // 搜索什么
   );
```

搜寻短整型数组

```
int  ArrayBsearch(
   const short&    array[],   // 用来搜索的数组
   short           value      // 搜索什么
   );
```

搜寻字符型数组

```
int  ArrayBsearch(
   const char&    array[],   // 用来搜索的数组
   char           value      // 搜索什么
   );
```

参量

*array[]*

[in] 搜寻数字数组。

*value*

[in] 搜寻值

© 2000-2025, MetaQuotes Ltd.

返回值

函数返回创立元素索引，如果需要值没找到，函数返回最相近值。

注释

二进位检索过程只在存储数组中进行，使用 ArraySort() 函数分类数字数组。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "Script based on RSI indicator data displays"
#property description "how often the market was in"
#property description "overbought and oversold areas in the specified time interval."
//--- 启动脚本时显示导入参数的窗口
#property script_show_inputs
//--- 导入参数
input int                InpMAPeriod=14;                    // 移动平均周期
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;       // 价格类型
input double             InpOversoldValue=30.0;             // 超卖水平
input double             InpOverboughtValue=70.0;           // 超买水平
input datetime           InpDateStart=D'2012.01.01 00:00';  // 分析开始日期
input datetime           InpDateFinish=D'2013.01.01 00:00'; // 分析结束日期
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   double rsi_buff[]; // 指标值的数组
   int    size=0;     // 数组大小
//--- 接收RSI指标处理
   ResetLastError();
   int rsi_handle=iRSI(Symbol(),Period(),InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- 接收指标处理失败
      PrintFormat("Indicator handle receiving error. Error code = %d",GetLastError());
      return;
     }
//--- 循环中，直到指标计算它的所有值
   while(BarsCalculated(rsi_handle)==-1)
     {
      //--- 如果指标强制完成脚本操作则退出
      if(IsStopped())
         return;
      //--- 暂停允许指标计算它的所有值
      Sleep(10);
     }
//--- 复制一段时间的指标值
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,InpDateFinish,rsi_buff)==-1)
     {
      PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError
      return;
     }
//--- 接收数组大小
   size=ArraySize(rsi_buff);
//--- 分类数组
   ArraySort(rsi_buff);
//--- 找出市场在超卖区域的时间 佰分比）
   double ovs=(double)ArrayBsearch(rsi_buff,InpOversoldValue)*100/(double)size;
//--- 找出市场在超买区域的时间 佰分比）
   double ovb=(double)(size-ArrayBsearch(rsi_buff,InpOverboughtValue))*100/(double)siz
//--- 显示数据的字符串形式
   string str="From "+TimeToString(InpDateStart,TIME_DATE)+" to "
              +TimeToString(InpDateFinish,TIME_DATE)+" the market was:";
   string str_ovb="in overbought area "+DoubleToString(ovb,2)+"% of time";
   string str_ovs="in oversold area "+DoubleToString(ovs,2)+"% of time";
//--- 在图表上显示数据
   CreateLabel("top",5,60,str,clrDodgerBlue);
   CreateLabel("overbought",5,35,str_ovb,clrDodgerBlue);
   CreateLabel("oversold",5,10,str_ovs,clrDodgerBlue);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重绘图表
   ChartRedraw(0);
//--- 暂停
   Sleep(10000);
  }
//+------------------------------------------------------------------+
//| 在图表左下角显示评论                                             |
//+------------------------------------------------------------------+
void CreateLabel(const string name,const int x,const int y,
                 const string str,const color clr)
  {
//--- 创建一个标签
   ObjectCreate(0,name,OBJ_LABEL,0,0,0);
//--- 绑定标签到左下角
   ObjectSetInteger(0,name,OBJPROP_CORNER,CORNER_LEFT_LOWER);
//--- 改变定位点位置
   ObjectSetInteger(0,name,OBJPROP_ANCHOR,ANCHOR_LEFT_LOWER);
//--- X方向定位点的距离
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
//--- Y方向定位点的距离
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
//--- 标签文本
   ObjectSetString(0,name,OBJPROP_TEXT,str);
//--- 文本颜色
   ObjectSetInteger(0,name,OBJPROP_COLOR,clr);
//--- 文本大小
   ObjectSetInteger(0,name,OBJPROP_FONTSIZE,12);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayCopy

复制一组数组到另一组。

```
int  ArrayCopy(
   void&         dst_array[],          // 目标数组
   const void&   src_array[],          // 源数组
   int           dst_start=0,          // 写入目标数组的指数
   int           src_start=0,          // 源数组的最初指数
   int           count=WHOLE_ARRAY     // 元素数量
   );
```

参量

*dst_array[]*

[out] 目标数组

*src_array[]*

[in] 源数组

*dst_start=0*

[in] 从目标数组的第几位开始写入，默认为0。

*src_start=0*

[in] 从源数组的第几位开始读取，默认为0。

*count=WHOLE_ARRAY*

[in] 复制数组的数量，默认全部数组复制 (count=WHOLE_ARRAY).

返回值

返回复制元素的数量

注释

如果count<0 or count>src_size-src_start,所有余下的数组部分都被复制，数组从左至右复制，对于一系列数组来说。

如果数组存在不同种类型，在复制期间试着改变源数组的每个元素到目标数组。字符串型数组只可以复制到字符串型数组中间，类和结构数组包括的需要初始化的对象不能复制。结构的一组数组只可以复制成相同类型数组。

对于如在时间序列索引的动态数组，目标数组的大小自动增加到复制数据的数量 (如果后者超过数组大小)。目标数组大小不会自动减少。

示例：

```
#property description "The indicator highlights the candlesticks that are local"
#property description "highs and lows. Interval length for finding"
#property description "extreme values should be found using an input parameters."
//--- 指标设置
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
```

© 2000-2025, MetaQuotes Ltd.

```
//---- 绘制
#property indicator_label1  "Extremums"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrLightSteelBlue,clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 预定义常量
#define INDICATOR_EMPTY_VALUE 0.0
//--- 输入参数
input int InpNum=4; // 半区间长度
//--- 指标缓冲区
double ExtOpen[];
double ExtHigh[];
double ExtLow[];
double ExtClose[];
double ExtColor[];
//--- 全局变量
int    ExtStart=0; // 第一个蜡烛图的指数不是一个极值
int    ExtCount=0; // 区间的非极值数量
//+------------------------------------------------------------------+
//| 填写非极值蜡烛图                                                 |
//+------------------------------------------------------------------+
void FillCandles(const double &open[],const double &high[],
                 const double &low[],const double &close[])
  {
//--- 填写蜡烛图
   ArrayCopy(ExtOpen,open,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtHigh,high,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtLow,low,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtClose,close,ExtStart,ExtStart,ExtCount);
  }
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓存映射
   SetIndexBuffer(0,ExtOpen);
   SetIndexBuffer(1,ExtHigh);
   SetIndexBuffer(2,ExtLow);
   SetIndexBuffer(3,ExtClose);
   SetIndexBuffer(4,ExtColor,INDICATOR_COLOR_INDEX);
//--- 指定值，不显示
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- 指定用于在数据窗口显示的指标缓存名称
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   return(INIT_SUCCEEDED);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 自定义指标的迭代函数                                             |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 在时间序列设置直接索引
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- 柱形变量计算开始
   int start=prev_calculated;
//--- 开始的InpNum*2柱不执行计算
   if(start==0)
     {
      start+=InpNum*2;
      ExtStart=0;
      ExtCount=0;
     }
//--- 如果只形成柱，检查下一个潜在极值
   if(rates_total-start==1)
      start--;
//--- 检查极值的柱形索引
   int ext;
//--- 指标值计算循环
   for(int i=start;i<rates_total-1;i++)
     {
      //--- 在i柱上无初始绘制
      ExtOpen[i]=0;
      ExtHigh[i]=0;
      ExtLow[i]=0;
      ExtClose[i]=0;
      //--- 用于检查的极值指数
      ext=i-InpNum;
      //--- 检查本地最大值
      if(IsMax(high,ext))
        {
         //--- 强调极值蜡烛图
         ExtOpen[ext]=open[ext];
         ExtHigh[ext]=high[ext];
```

© 2000-2025, MetaQuotes Ltd.

```
         ExtLow[ext]=low[ext];
         ExtClose[ext]=close[ext];
         ExtColor[ext]=1;
         //--- 用中间色突出其他蜡烛图直至极值
         FillCandles(open,high,low,close);
         //--- 改变变量颜色
         ExtStart=ext+1;
         ExtCount=0;
         //--- 传递到下一个迭代
         continue;
        }
      //--- 检查本地最小值
      if(IsMin(low,ext))
        {
         //--- 突出极值蜡烛图
         ExtOpen[ext]=open[ext];
         ExtHigh[ext]=high[ext];
         ExtLow[ext]=low[ext];
         ExtClose[ext]=close[ext];
         ExtColor[ext]=2;
         //--- 用中间色突出其他蜡烛图直至极值
         FillCandles(open,high,low,close);
         //--- 改变变量值
         ExtStart=ext+1;
         ExtCount=0;
         //--- 传递到下一个迭代
         continue;
        }
      //--- 增加无极值间隔数量
      ExtCount++;
     }
//--- 为下次调用返回 prev_calculated 值
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 检查当前数组元素是否本地最高                                     |
//+------------------------------------------------------------------+
bool IsMax(const double &price[],const int ind)
  {
//--- 间隔开始变量
   int i=ind-InpNum;
//--- 间隔时间结束
   int finish=ind+InpNum+1;
//--- 检查上半部间隔
   for(;i<ind;i++)
     {
      if(price[ind]<=price[i])
         return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 检查下半部间隔
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]<=price[i])
         return(false);
     }
//--- 这是一个极值
   return(true);
  }
//+------------------------------------------------------------------+
//| 检查当前数组元素是否是本地最低                                   |
//+------------------------------------------------------------------+
bool IsMin(const double &price[],const int ind)
  {
//--- 间隔开始变量
   int i=ind-InpNum;
//--- 间隔时间结束
   int finish=ind+InpNum+1;
//--- 检查上半部间隔
   for(;i<ind;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- 检查下半部间隔
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- 这是一个极值
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayCompare

函数返回相同类型两个数组的比较结果。它可以用来比较简单类型或没有 复杂对象的自定义结构的数组，自定义结构不可以包括 字符串，动态数组，类和其他没有复杂对象的结构。

```
int  ArrayCompare(
   const void&  array1[],            // 第一数组
   const void&  array2[],            // 第二数组
   uint         start1=0,            // 第一数组的初始偏差
   uint         start2=0,            // 第二数组的初始偏差
   uint         count=WHOLE_ARRAY    // 元素比较的数量
   );
```

参数

*array1[]*

[in] 第一数组。

*array2[]*

[in] 第二数组。

*start1=0*

[in] 第一数组的元素初始指数，比较的开始。默认开始指数 - 0。

*start2=0*

[in] 第二数组的元素初始指数，比较的开始。默认开始指数 - 0。

*count=WHOLE_ARRAY*

[in] 要比较的元素数量。两种数组的所有元素默认都参与比较 (count=WHOLE_ARRAY)。

返回值

- -1, 如果 array1[] 小于 array2[]
- 0, 如果 array1[] 等于 array2[]
- 1, 如果 array1[] 大于 array2[]
- -2, 如果由于比较的数组类型不一致而产生错误或如果start1, start2或计算值导致外数组降低。

注意

在数组忠于另一个数组的情况下，如果数组在大小和count=WHOLE_ARRAY不相同，那么函数不会返回0 (数组将不会认为相等)。在这种情况下，比较数组大小的结果将返回： -1，如果array1[]大小小于array2[]大小，否则为1。

示例：

```
//--- 全局变量
double   ExtArrayFirst[];
double   ExtArraySecond[];

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置数组大小
   if(ArrayResize(ExtArrayFirst,10)!=10)
     {
      Print("ArrayResize() failed for ExtArrayFirst. Error code: ",GetLastError());
      return;
     }
   if(ArrayResize(ExtArraySecond,10)!=10)
     {
      Print("ArrayResize() failed for ExtArraySecond. Error code: ",GetLastError());
      return;
     }

//--- 在循环中用i和j索引的值填充数组
   int total=ArraySize(ExtArrayFirst);
   for(int i=0, j=total-1; i<total; i++,j--)
     {
      //--- 从左向右填充ExtArrayFirst数组
      //--- 从右向左填充ExtArraySecond数组
      ExtArrayFirst[i]=i;
      ExtArraySecond[i]=j;
     }
//--- 比较数组并在日志中记录结果
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
  结果:
   ExtArrayFirst:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   ExtArraySecond:
   9.00000 8.00000 7.00000 6.00000 5.00000 4.00000 3.00000 2.00000 1.00000 0.00000
   Result ArrayCompare(): ExtArrayFirst is smaller than ExtArraySecond (result = -1)
   */

//--- 现在让我们来浏览数组
//--- 在循环中用i和j索引的值填充数组
   for(int i=0, j=total-1; i<total; i++,j--)
     {
      //--- 从右向左填充ExtArrayFirst数组
      //--- 从左向右填充ExtArraySecond数组
      ExtArrayFirst[i]=j;
      ExtArraySecond[i]=i;
     }
//--- 比较数组并在日志中记录结果
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
  结果:
   ExtArrayFirst:
```

© 2000-2025, MetaQuotes Ltd.

```
   9.00000 8.00000 7.00000 6.00000 5.00000 4.00000 3.00000 2.00000 1.00000 0.00000
   ExtArraySecond:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   Result ArrayCompare(): ExtArrayFirst is larger than ExtArraySecond (result = 1)
   */

//--- 现在让我们沿一个方向填充数组
//--- 在循环中用i索引值填充数组
   for(int i=0; i<total; i++)
     {
      //--- fill both arrays from left to right
      ExtArrayFirst[i]=i;
      ExtArraySecond[i]=i;
     }
//--- 比较数组并在日志中记录结果
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
  结果：
   ExtArrayFirst:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   ExtArraySecond:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   Result ArrayCompare(): ExtArrayFirst and ExtArraySecond are equal (result = 0)
   */
  }
//+------------------------------------------------------------------+
//| 比较和显示结果                                                   |
//+------------------------------------------------------------------+
void ArrayComparePrint(const double &array1[], const double &array2[])
  {
   //--- 打印数组的标题和内容
   Print("ExtArrayFirst:");
   ArrayPrint(array1);
   Print("ExtArraySecond:");
   ArrayPrint(array2);
   //--- 比较数组并输出比较结果
   int    res=ArrayCompare(array1,array2);
   string res_str=(res>0 ? "ExtArrayFirst is larger than ExtArraySecond" : res<0 ? "Ex
   PrintFormat("Result ArrayCompare(): %s (result = %d)\n",res_str,res);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# ArrayFree

任何动态数组会空出来缓冲区并建立0维大小

```
void  ArrayFree(
   void&  array[]      // 数组
   );
```

参量

*array[]*

[in] 动态数组。

返回值

没有返回值

注释

考虑到要一次性释放所有使用的内存并且数组的主要工作由访问指标缓存组成，那么使用ArrayFree() 函数的需求不会经常出现。缓存大小自动由程序端的执行子系统管理。

如果在应用程序复杂的动态环境中需要手动管理内存，ArrayFree() 函数会允许用户立即明确地释放已经不需要的动态数组占用的内存。

示例：

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- 预定义常量
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+------------------------------------------------------------------+
//| 内存工作的对话类                                                 |
//+------------------------------------------------------------------+
class CMemoryControl : public CAppDialog
  {
private:
   //--- 数组大小
   int               m_arr_size;
   //--- 数组
   char              m_arr_char[];
   int               m_arr_int[];
   float             m_arr_float[];
   double            m_arr_double[];
   long              m_arr_long[];
   //--- 标签
   CLabel            m_lbl_memory_physical;
```

© 2000-2025, MetaQuotes Ltd.

```
   CLabel            m_lbl_memory_total;
   CLabel            m_lbl_memory_available;
   CLabel            m_lbl_memory_used;
   CLabel            m_lbl_array_size;
   CLabel            m_lbl_array_type;
   CLabel            m_lbl_error;
   CLabel            m_lbl_change_type;
   CLabel            m_lbl_add_size;
   //--- 按钮
   CButton           m_button_add;
   CButton           m_button_free;
   //--- 组合框
   CComboBox         m_combo_box_step;
   CComboBox         m_combo_box_type;
   //--- 来自组合框的当前的数组类型值
   int               m_combo_box_type_value;

public:
                     CMemoryControl(void);
                    ~CMemoryControl(void);
   //--- 类对象创建方法
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- 图表事件处理程序
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- 创建标签
   bool              CreateLabel(CLabel &lbl,const string name,const int x,const int y
   //--- 创建控制元素
   bool              CreateButton(CButton &button,const string name,const int x,const
   bool              CreateComboBoxStep(void);
   bool              CreateComboBoxType(void);
   //--- 事件处理程序
   void              OnClickButtonAdd(void);
   void              OnClickButtonFree(void);
   void              OnChangeComboBoxType(void);
   //--- 当前数组工作的方法
   void              CurrentArrayFree(void);
   bool              CurrentArrayAdd(void);
  };
//+------------------------------------------------------------------+
//| 释放当前数组内存                                                 |
//+------------------------------------------------------------------+
void CMemoryControl::CurrentArrayFree(void)
  {
//--- 重置数组大小
   m_arr_size=0;
//--- 释放数组
   if(m_combo_box_type_value==0)
```

© 2000-2025, MetaQuotes Ltd.

```
      ArrayFree(m_arr_char);
   if(m_combo_box_type_value==1)
      ArrayFree(m_arr_int);
   if(m_combo_box_type_value==2)
      ArrayFree(m_arr_float);
   if(m_combo_box_type_value==3)
      ArrayFree(m_arr_double);
   if(m_combo_box_type_value==4)
      ArrayFree(m_arr_long);
  }
//+------------------------------------------------------------------+
//| 尝试为当前数组增加内存                                           |
//+------------------------------------------------------------------+
bool CMemoryControl::CurrentArrayAdd(void)
  {
//--- 如果所用内存大小超过物理内存大小，则退出
   if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TERMINAL_MEMOR
      return(false);
//--- 根据当前类型尝试分配内存
   if(m_combo_box_type_value==0 && ArrayResize(m_arr_char,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==1 && ArrayResize(m_arr_int,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==2 && ArrayResize(m_arr_float,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==3 && ArrayResize(m_arr_double,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==4 && ArrayResize(m_arr_long,m_arr_size)==-1)
      return(false);
//--- 分配内存
   return(true);
  }
//+------------------------------------------------------------------+
//| 事件处理                                                         |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CMemoryControl)
ON_EVENT(ON_CLICK,m_button_add,OnClickButtonAdd)
ON_EVENT(ON_CLICK,m_button_free,OnClickButtonFree)
ON_EVENT(ON_CHANGE,m_combo_box_type,OnChangeComboBoxType)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| 构造函数                                                         |
//+------------------------------------------------------------------+
CMemoryControl::CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| 析构函数                                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
CMemoryControl::~CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| 类对象创建方法                                                   |
//+------------------------------------------------------------------+
bool CMemoryControl::Create(const long chart,const string name,const int subwin,
                            const int x1,const int y1,const int x2,const int y2)
  {
//--- 创建基本类对象
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- 准备标签字符串
   string str_physical="Memory physical = "+(string)TerminalInfoInteger(TERMINAL_MEMOF
   string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_TOTA
   string str_available="Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEM
   string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED)+
//--- 创建标签
   if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_START+5,str_phy
      return(false);
   if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,str_total,12
      return(false);
   if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_START+55,str_
      return(false);
   if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,str_used,12,cl
      return(false);
   if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"Array type =
      return(false);
   if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"Array size =
      return(false);
   if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,clrRed))
      return(false);
   if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_START+185,"Chang
      return(false);
   if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,"Add to arra
      return(false);
//--- 创建控制元素
   if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add",12,clrBlue)
      return(false);
   if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Free",12,clrBl
      return(false);
   if(!CreateComboBoxType())
      return(false);
   if(!CreateComboBoxStep())
      return(false);
//--- 初始变量
   m_arr_size=0;
//--- 成功执行
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 创建按钮                                                         |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateButton(CButton &button,const string name,const int x,
                                  const int y,const string str,const int font_size,
                                  const int clr)
  {
//--- 创建按钮
   if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
      return(false);
//--- 文本
   if(!button.Text(str))
      return(false);
//--- 字体大小
   if(!button.FontSize(font_size))
      return(false);
//--- 标签颜色
   if(!button.Color(clr))
      return(false);
//--- 添加按钮到控制元素
   if(!Add(button))
      return(false);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建数组大小组合框                                               |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxStep(void)
  {
//--- 创建组合框
   if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START+100,Y_START
      return(false);
//--- 添加元素到组合框
   if(!m_combo_box_step.ItemAdd("100 000",100000))
      return(false);
   if(!m_combo_box_step.ItemAdd("1 000 000",1000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("10 000 000",10000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("100 000 000",100000000))
      return(false);
//--- 设置当前组合框元素
   if(!m_combo_box_step.SelectByValue(1000000))
      return(false);
//--- 添加组合框到控制元素
   if(!Add(m_combo_box_step))
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建数组类型组合框                                               |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxType(void)
  {
//--- 创建组合框
   if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START+100,Y_START
      return(false);
//--- 添加元素到组合框
   if(!m_combo_box_type.ItemAdd("char",0))
      return(false);
   if(!m_combo_box_type.ItemAdd("int",1))
      return(false);
   if(!m_combo_box_type.ItemAdd("float",2))
      return(false);
   if(!m_combo_box_type.ItemAdd("double",3))
      return(false);
   if(!m_combo_box_type.ItemAdd("long",4))
      return(false);
//--- 设置当前组合框元素
   if(!m_combo_box_type.SelectByValue(3))
      return(false);
//--- 存储当前组合框元素
   m_combo_box_type_value=3;
//--- 添加组合框到控制元素
   if(!Add(m_combo_box_type))
      return(false);
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建标签                                                         |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x,
                                 const int y,const string str,const int font_size,
                                 const int clr)
  {
//--- 创建标签
   if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
      return(false);
//--- 文本
   if(!lbl.Text(str))
      return(false);
//--- 字体大小
   if(!lbl.FontSize(font_size))
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 颜色
   if(!lbl.Color(clr))
      return(false);
//--- 添加标签到控制元素
   if(!Add(lbl))
      return(false);
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 点击 "添加"按钮事件的处理程序                                    |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonAdd(void)
  {
//--- 提高数组大小
   m_arr_size+=(int)m_combo_box_step.Value();
//--- 尝试为当前数组分配内存
   if(CurrentArrayAdd())
     {
      //--- 分配内存，在屏幕上展示当前状态
      m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TE
      m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEM
      m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
      m_lbl_error.Text("");
     }
   else
     {
      //--- 分配内存失败，显示错误信息
      m_lbl_error.Text("Array is too large, error!");
      //--- 返回之前的数组大小
      m_arr_size-=(int)m_combo_box_step.Value();
     }
  }
//+------------------------------------------------------------------+
//| 点击 "释放"按钮事件的处理程序                                    |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonFree(void)
  {
//--- 释放当前数组的内存
   CurrentArrayFree();
//--- 在屏幕上显示当前状态
   m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERMI
   m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY
   m_lbl_array_size.Text("Array size = 0");
   m_lbl_error.Text("");
  }
//+------------------------------------------------------------------+
//| 组合框更改事件的处理程序                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void CMemoryControl::OnChangeComboBoxType(void)
  {
//--- 检查数组类型是否更改
   if(m_combo_box_type.Value()!=m_combo_box_type_value)
     {
      //--- 释放当前数组内存
      OnClickButtonFree();
      //--- 使用另一个数组类型
      m_combo_box_type_value=(int)m_combo_box_type.Value();
      //--- 在屏幕上显示新数组类型
      if(m_combo_box_type_value==0)
         m_lbl_array_type.Text("Array type = char");
      if(m_combo_box_type_value==1)
         m_lbl_array_type.Text("Array type = int");
      if(m_combo_box_type_value==2)
         m_lbl_array_type.Text("Array type = float");
      if(m_combo_box_type_value==3)
         m_lbl_array_type.Text("Array type = double");
      if(m_combo_box_type_value==4)
         m_lbl_array_type.Text("Array type = long");
     }
  }
//--- CMemoryControl 类对象
CMemoryControl ExtDialog;
//+------------------------------------------------------------------+
//| 专家初始化类型                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建对话
   if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
      return(INIT_FAILED);
//--- 启动
   ExtDialog.Run();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| 专家图表事件函数                                                 |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
```

© 2000-2025, MetaQuotes Ltd.

```
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
 ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

© 2000-2025, MetaQuotes Ltd.

# ArrayGetAsSeries

检测数组标引的导向

```
bool  ArrayGetAsSeries(
   const void&  array[]    // 用于检测的数组
   );
```

参量

*array[]*

[in] 检测数组

返回值

返回 true，如果规定数组建立了AS_SERIES 标志，如下，访问数组以时序列无序执行，一个 时间序列 与普通数组通常在从未到头 (从最新数据到最老)执行时序列元素索引不同。

注释

检验数组是否属于时序列，使用 ArrayIsSeries() 函数。价格数组通过输入参量无需强制传递到 OnCalculate()函数中。必要的搜索导向可以通过使用 ArraySetAsSeries() 函数建立。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "Indicator calculates absolute values of the difference between"
#property description "Open and Close or High and Low prices displaying them in a sepa
#property description "as a histogram."
//---指标设置
#property indicator_separate_window
#property indicator_buffers  1
#property indicator_plots    1
//----- 绘制
#property indicator_type1    DRAW_HISTOGRAM
#property indicator_style1   STYLE_SOLID
#property indicator_width1   3
//--- 导入参数
input bool InpAsSeries=true; // 指标缓冲区的标引导向
input bool InpPrices=true;   // 计算价格 (true - 开盘价,收盘价; false - 最高价,最低价)
//--- 指标缓冲区
double ExtBuffer[];
//+------------------------------------------------------------------+
//| 计算指标的值                                                     |
//+------------------------------------------------------------------+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                        const double &first[],const double &second[],double &buffer[])
  {
//--- 计算柱形的开始变量
   int start=prev_calculated;
//--- 如果前一个订单号已经计算了指标值,那么使用最近的柱
   if(prev_calculated>0)
      start--;
//--- 定义数组的标引导向
   bool as_series_first=ArrayGetAsSeries(first);
   bool as_series_second=ArrayGetAsSeries(second);
   bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- 如果有必要,直接替代标引导向
   if(as_series_first)
      ArraySetAsSeries(first,false);
   if(as_series_second)
      ArraySetAsSeries(second,false);
   if(as_series_buffer)
      ArraySetAsSeries(buffer,false);
//--- 计算指标值
   for(int i=start;i<rates_total;i++)
      buffer[i]=MathAbs(first[i]-second[i]);
  }
//+------------------------------------------------------------------+
//| 自定义指标初始函数                                               |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定指标缓冲区
   SetIndexBuffer(0,ExtBuffer);
//--- 在指标缓冲区设置标引元素
   ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- 检查指标计算的价格
   if(InpPrices)
     {
      //--- 开盘价和收盘价
      PlotIndexSetString(0,PLOT_LABEL,"BodySize");
      //--- 设置指标颜色
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrOrange);
     }
   else
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 最高价和最低价
      PlotIndexSetString(0,PLOT_LABEL,"ShadowSize");
      //--- 设置指标颜色
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrDodgerBlue);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 根据标识值计算指标
   if(InpPrices)
      CandleSizeOnBuffer(rates_total,prev_calculated,open,close,ExtBuffer);
   else
      CandleSizeOnBuffer(rates_total,prev_calculated,high,low,ExtBuffer);
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

另见

接入时间序列，ArraySetAsSeries

© 2000-2025, MetaQuotes Ltd.

# ArrayInitialize

函数通过预设值初始化一组数字数组。

初始化字符型数组

```
int  ArrayInitialize(
   char     array[],      // 初始化的数组
   char     value         // 将被设置的值
   );
```

初始化短整型数组

```
int  ArrayInitialize(
   short    array[],      // 初始化的数组
   short    value         // 将被设置的值
   );
```

初始化整型数组

```
int  ArrayInitialize(
   int      array[],      // 初始化的数组
   int      value         // 将被设置的值
   );
```

初始化长整型数组

```
int  ArrayInitialize(
   long     array[],      // 初始化的数组
   long     value         // 将被设置的值
   );
```

初始化浮点型数组

```
int  ArrayInitialize(
   float    array[],      // 初始化的数组
   float    value         // 将被设置的值
   );
```

初始化双精度型数组

```
int  ArrayInitialize(
   double   array[],      // 初始化的数组
   double   value         // 将被设置的值
   );
```

初始化布尔型数组

```
int  ArrayInitialize(
   bool     array[],      // 初始化的数组
   bool     value         // 将被设置的值
   );
```

© 2000-2025, MetaQuotes Ltd.

初始化无符号整型数组

```
int  ArrayInitialize(
   uint    array[],      // 初始化的数组
   uint    value         // 将被设置的值
   );
```

参量

*array[]*

[out] 需要初始化的数值数组

*value*

[in] 为所有数组元素建立新值

返回值

初始化元素的数量。

注释

ArrayResize()函数允许建立为无需人工安置扩散而保留的数组内存大小。它为了更好的属性而实施，因为内存重新安置操作相当慢。

使用ArrayInitialize(数组, init_val)初始化不表示储备元素相同值分配给数组。使用ArrayResize()函数进行*数组*的长远扩展，元素会添加到数组末尾，其他值是未下定义的病情在大多数情况下不等于*init_value*。

示例：

```
void OnStart()
  {
//--- 动态数组
   double array[];
//--- 数组大小设为100个元素并为另10个元素保留缓冲区
   ArrayResize(array,100,10);
//--- 初始化带有EMPTY_VALUE=DBL_MAX值的数组元素
   ArrayInitialize(array,EMPTY_VALUE);
   Print("Values of 10 last elements after initialization");
   for(int i=90;i<100;i++) printf("array[%d] = %G",i,array[i]);
//--- 数组扩展5个元素
   ArrayResize(array,105);
   Print("Values of 10 last elements after ArrayResize(array,105)");
//--- 最后的5个数组值可以从保留的缓冲区获得
   for(int i=95;i<105;i++) printf("array[%d] = %G",i,array[i]);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayFill

以指定值填充数组的函数。

```
void  ArrayFill(
   void&  array[],        // 数组
   uint   start,          // 开始标引
   uint   count,          // 填充的元素数量
   void   value           // 值
   );
```

参数

*array[]*

[out] 简单类型的数组 (char, uchar, short, ushort, int, uint, long, ulong, bool, color, datetime, float, double)。

*start*

[in] 开始标引。在这种情况下，指定的AS_SERIES flag被忽略。

*count*

[in] 填充的元素数量。

*value*

[in] 填充数组的值。

返回值

无返回值。

注意

当调用ArrayFill()函数时，正常的标引导向（从左至右）始终隐藏。它意味着改变使用ArraySetAsSeries()函数访问数组元素的顺序被忽略。

当通过ArrayFill()函数处理时多维数组显示为一维数组。例如，array[2][4] 处理为array[8]。因此，当使用该数组时，您可以指定初始元素的标引等于5。因此，为 array[2][4] 调用ArrayFill(array, 5, 2, 3.14)，用3.14来填充array[1][1] 和 array[1][2]。

例如：

```
void OnStart()
  {
//--- 声明动态数组
   int a[];
//--- 设置大小
   ArrayResize(a,10);
//--- 用123填充前5个元素
   ArrayFill(a,0,5,123);
//--- 用456填充后5个元素
   ArrayFill(a,5,5,456);
//--- 显示值
   for(int i=0;i<ArraySize(a);i++) printf("a[%d] = %d",i,a[i]);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayIsDynamic

判断动态数组的函数。

```
bool  ArrayIsDynamic(
   const void&  array[]    // 已检测的数组
   );
```

参量

*array[]*

[in] 检测数组.

返回值

数组是动态，返回 true，否则返回 false。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "This indicator does not calculate values. It makes a single att
#property description "apply the call of ArrayFree() function to three arrays: dynamic
#property description "an indicator buffer. Results are shown in Experts journal."
//--- 指标设置
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 全局变量
double ExtDynamic[];   // 动态数组
double ExtStatic[100]; // 静态数组
bool   ExtFlag=true;   // 标识
double ExtBuff[];      // 指标缓冲区
//+------------------------------------------------------------------+
//| 自定义指标初始函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 为数组分配内存
   ArrayResize(ExtDynamic,100);
//--- 指标缓冲区映射
   SetIndexBuffer(0,ExtBuff);
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])
  {
//--- 执行一个分析
   if(ExtFlag)
     {
      //--- 尝试释放数组内存
      //--- 1. 动态数组
      Print("+============================+");
      Print("1. Check dynamic array:");
      Print("Size before memory is freed = ",ArraySize(ExtDynamic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtDynamic) ? "Yes" : "No");
      //--- 尝试释放数组内存
      ArrayFree(ExtDynamic);
      Print("Size after memory is freed = ",ArraySize(ExtDynamic));
      //--- 2. 静态数组
      Print("2. Check static array:");
      Print("Size before memory is freed = ",ArraySize(ExtStatic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtStatic) ? "Yes" : "No");
      //--- 尝试释放数组内存
      ArrayFree(ExtStatic);
      Print("Size after memory is freed = ",ArraySize(ExtStatic));
      //--- 3. 指标缓冲区
      Print("3. Check indicator buffer:");
      Print("Size before memory is freed = ",ArraySize(ExtBuff));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtBuff) ? "Yes" : "No");
      //--- 尝试释放数组内存
      ArrayFree(ExtBuff);
      Print("Size after memory is freed = ",ArraySize(ExtBuff));
      //--- 改变标识的值
      ExtFlag=false;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

另见

接入时间序列和指标

© 2000-2025, MetaQuotes Ltd.

# ArrayIsSeries

检测时间序列数组的函数。

```
bool  ArrayIsSeries(
   const void&  array[]    // 已经检测的数组
   );
```

参量

*array[]*

[in] 检测数组

返回值

如果检测数组是时序列返回真值，否则返回错误值。数组以参量形式传递到OnCalculate()函数一定通过 ArrayGetAsSeries()检测访问数组元素命令

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 标签图1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标缓冲区
double         Label1Buffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(ArrayIsSeries(open))
      Print("open[] is timeseries");
   else
      Print("open[] is not timeseries!!!");
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

另见

接入时间序列和指标

© 2000-2025, MetaQuotes Ltd.

# ArrayMaximum

搜索多维度数值数组的第一维度中的最大元素。

```
int  ArrayMaximum(
   const void&   array[],              // 用于搜索的数组
   int           start=0,              // 启动用于检测的指标
   int           count=WHOLE_ARRAY     // 检测元素数量
   );
```

参量

*array[]*

[in] 数字数组，建立搜寻

*start=0*

[in] 开始符合性搜索

*count=WHOLE_ARRAY*

[in] 搜索元素的数量。默认在整个数组 (count=WHOLE_ARRAY)中搜索。

返回值

函数返回发现元素的标引索引，考虑数组系列，失败返回-1

注释

搜索最大值时考虑AS_SERIES标识值。

函数ArrayMaximum 和 ArrayMinimum 接受任何维度的数组为参数。然而，搜索却始终应用于第一(0)维度。

示例：

```
#property description "The indicator displays larger time frame's candlesticks on the
//--- 指标设置
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- 图 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- 图 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- 图 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- 图 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
//---- 图 5
#property indicator_label5  "BullBody"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color5  clrOlive,clrOlive
//---- 图 6
#property indicator_label6  "BullBodyEnd"
#property indicator_color6  clrOlive,clrOlive
//---- 图 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- 图 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- 预定义常量
#define INDICATOR_EMPTY_VALUE 0.0
//--- 输入参数
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // 指标计算的时间范围
input datetime        InpDateStart=D'2013.01.01 00:00'; // 分析起始日期
//--- 熊市蜡烛图的指标缓冲区
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- 牛市蜡烛图的指标缓冲区
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- 全局变量
datetime ExtTimeBuff[];      // 较大时间帧的时间缓冲区
int      ExtSize=0;          // 时间缓冲区大小
int      ExtCount=0;         // 时间缓冲区的标引
int      ExtStartPos=0;      // 指标计算的初始位置
bool     ExtStartFlag=true;  // 接收初始位置的辅助标识
datetime ExtCurrentTime[1];  // 生成较大时间帧柱形的最近时间
datetime ExtLastTime;        // 较大时间帧的最近时间，执行计算
bool     ExtBearFlag=true;   // 标记定义编写数据订单熊市指标缓冲区
bool     ExtBullFlag=true;   // 标记定义编写数据订单牛市指标缓冲区
int      ExtIndexMax=0;      // 数组中最大元素标引
int      ExtIndexMin=0;      // 数组中最小元素标引
int      ExtDirectionFlag=0; // 当前蜡烛图的价格移动方向
//--- 正确绘制蜡烛图开盘价和收盘价之间的转换
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 填写蜡烛图的基本部分                                                |
//+------------------------------------------------------------------+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- 在数组中找出最大和最小元素的标引
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // 最高极值
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // 最低极值
//--- 定义当前时间帧填充多少柱
   int count=fill_index-start+1;
//--- 如果首柱的收盘价超过最后的柱，那么蜡烛图为熊市
   if(open[start]>close[last])
     {
      //--- 如果蜡烛图之前牛市，清空牛市指标缓冲区的值
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
      //--- 熊市蜡烛图
      ExtDirectionFlag=-1;
      //--- 生成蜡烛图
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     close[last],high[index_max],low[index_min],start,count,ExtBearFla
      //--- 退出函数
      return;
     }
//--- 如果首柱的收盘价少于最后的柱，那么蜡烛图为牛市
   if(open[start]<close[last])
     {
      //--- 如果蜡烛图之前为熊市，清空熊市指标缓冲区的值
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
      //--- 牛市蜡烛图
      ExtDirectionFlag=1;
      //--- 生成蜡烛图
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
                     open[start],high[index_max],low[index_min],start,count,ExtBullFla
      //--- 退出函数
      return;
     }
//--- 如果您在这部分函数，首柱开盘价等于
//--- 最后柱的收盘价;这种蜡烛图被认为熊市图
//--- 如果在那之前蜡烛图为牛市，清空牛市指标缓冲区的值
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowS
//--- 熊市蜡烛图
   ExtDirectionFlag=-1;
//--- 如果收盘和开盘价相同，使用切换来正确显示
   if(high[index_max]!=low[index_min])
```

© 2000-2025, MetaQuotes Ltd.

```
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start],open[start]-ExtEmptyBodySize,high[index_max],
                     high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| 填充蜡烛图末端                                                   |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- 单柱时不要绘制
   if(last-start==0)
      return;
//--- 如果首柱的收盘价超过最后的柱，蜡烛图为熊市
   if(open[start]>close[last])
     {
      //--- 生成蜡烛图末端
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start],close[last],high[index_max],low[index_min],fill_index,
      //--- 退出函数
      return;
     }
//--- 如果首柱的收盘价小于最后的柱，蜡烛图为牛市
   if(open[start]<close[last])
     {
      //--- 生成蜡烛图末端
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ext
                    close[last],open[start],high[index_max],low[index_min],fill_index,
      //--- 退出函数
      return;
     }
//--- 如果您在这部分函数，首柱开盘价等于
//--- 最后柱的收盘价;这种蜡烛图被认为熊市图
//--- 生成蜡烛图末端
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_i
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEm
  }
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int OnInit()
  {
//--- 检查指标周期
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- 在前景显示价格数据
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- 绑定指标缓冲区
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- 设置一些属性值来创建指标
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // 图形结构类型
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // 绘制线型风格
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // 绘制线型宽度
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 如果没有计算的柱
```

© 2000-2025, MetaQuotes Ltd.

```
   if(prev_calculated==0)
     {
      //--- 接收较大时间帧柱形的到达时间
      if(!GetTimeData())
         return(0);
     }
//--- 设置直接索引
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- 启动计算柱形的变量
   int start=prev_calculated;
//--- 如果生成柱形，重新计算上面的指标值
   if(start!=0 && start==rates_total)
      start--;
//--- 循环计算指标值
   for(int i=start;i<rates_total;i++)
     {
      //--- 用空值填充指标缓冲区的i元素
      FillIndicatorBuffers(i);
      //--- 从InpDateStart日期开始计算柱形
      if(time[i]>=InpDateStart)
        {
         //--- 定义值第一时间显示的持仓
         if(ExtStartFlag)
           {
            //--- 存储最初柱形的数量
            ExtStartPos=i;
            //--- 定义超过time[i]的较大时间帧的初始日期
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- 改变标识值以便不在这模块再次运行
            ExtStartFlag=false;
           }
         //--- 检查数组中是否有任何元素
         if(ExtCount<ExtSize)
           {
            //--- 等候当前时间帧的值到达较大时间帧的值
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //--- 绘制蜡烛图的主要部分 不填写最后和倒数第二柱之间的区域）
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
               //--- 填充蜡烛图末端 最后和倒数第二柱之间的区域）
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
               //--- 切换初始持仓以绘制下一个蜡烛图
               ExtStartPos=i;
```

© 2000-2025, MetaQuotes Ltd.

```
                //--- 增加数组计数器
                ExtCount++;
               }
             else
                continue;
            }
          else
            {
             //--- 重置数组的值
             ResetLastError();
             //--- 接收来自较大时间帧的最后日期
             if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
               {
                Print("Data copy error, code = ",GetLastError());
                return(0);
               }
             //--- 如果新日期靠后，停止生成蜡烛图
             if(ExtCurrentTime[0]>ExtLastTime)
               {
                //--- 在主要指标缓冲区清空最后和倒数第二柱之间的区域
                ClearEndOfBodyMain(i-1);
                //--- 使用辅助指标缓冲区填充该区域
                FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
                //--- 切换初始持仓以绘制下一个蜡烛图
                ExtStartPos=i;
                //--- 重置价格方向标识
                ExtDirectionFlag=0;
                //--- 存储新的最近日期
                ExtLastTime=ExtCurrentTime[0];
               }
             else
               {
                //--- 生成蜡烛图
                FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndex
               }
            }
         }
      }
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 检查指定指标周期的正确性                                         |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
//--- 指标周期应该超过显示它的时间帧
   if(current_period>=high_period)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("Error! The value of the indicator period should exceed the value of the c
      return(false);
     }
//--- 如果指标周期是周或月份，周期则是正确的
   if(high_period>32768)
      return(true);
//--- 转变周期值到分钟
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- 指标周期应该是它显示的时间帧的几倍
   if(high_period%current_period!=0)
     {
      Print("Error! The value of the indicator period should be multiple of the value
      return(false);
     }
//--- 指标周期应该超过它显示的时间帧的3或更多倍
   if(high_period/current_period<3)
     {
      Print("Error! The indicator period should exceed the current time frame 3 or mo
      return(false);
     }
//--- 指标周期在当前时间帧是正确的
   return(true);
  }
//+------------------------------------------------------------------+
//| 接收来自较大时间帧的时间数据                                     |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- 重置错误的值
   ResetLastError();
//--- 复制当前时间的所有数据
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- 接收错误代码
      int code=GetLastError();
      //--- 打印错误信息
      PrintFormat("Data copy error! %s",code==4401
                  ? "History is still being uploaded!"
                  : "Code = "+IntegerToString(code));
      //--- 返回false，反复尝试下载数据
      return(false);
     }
//--- 接收数组大小
   ExtSize=ArraySize(ExtTimeBuff);
//--- 为数组设置循环指数为零
   ExtCount=0;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在时间帧设置当前蜡烛图持仓为零
   ExtStartPos=0;
   ExtStartFlag=true;
//--- 存储来自较大时间帧的最近时间值
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 函数形成了蜡烛图的主要部分。根据标识的                                          |
//| 值，函数定义了要使用哪个数据和数组                                              |
//| 以便正确显示                                                                  |
//+------------------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- 检查标识的值
   if(flag)
     {
      //--- 生成蜡烛图主体
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- 生成蜡烛图影子
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- 生成蜡烛图主体
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- 生成蜡烛图影子
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+------------------------------------------------------------------------------+
//| 函数形成了蜡烛图的末端。根据标识的值，                                          |
//| 函数定义了要使用哪个数据和数组                                                  |
//| 以便正确显示。                                                                |
//+------------------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
                   const int end,bool &flag)
  {
//--- 检查标识的值
   if(flag)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 生成蜡烛图主体的末端
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- 生成蜡烛图影子的末端
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- 改变标识值到相反方向
      flag=false;
     }
   else
     {
      //--- 生成蜡烛图主体的末端
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- 生成蜡烛图影子的末端
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- 改变标识值到相反方向
      flag=true;
     }
  }
//+------------------------------------------------------------------------------+
//| 清空蜡烛图末端 (最后和倒数第二柱之间的区域)                                    |
//|                                                                              |
//+------------------------------------------------------------------------------+
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecc
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecc
  }
//+------------------------------------------------------------------------------+
//| 清空蜡烛图                                                                   |
//+------------------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- 检查
   if(count!=0)
     {
      //--- 以空值填充指标缓冲区
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+------------------------------------------------------------------------------+
//| 生成蜡烛图的主要部分                                                         |
//+------------------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- 检查
```

© 2000-2025, MetaQuotes Ltd.

```
   if(count!=0)
     {
      //--- 用值填充指标缓冲区
      ArrayFill(fst,start,count,fst_value);
      ArrayFill(snd,start,count,snd_value);
     }
  }
//+------------------------------------------------------------------------------+
//| 生成蜡烛图末端                                                               |
//+------------------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
  {
//--- 用值填充指标缓冲区
   ArrayFill(fst,last-1,2,fst_value);
   ArrayFill(snd,last-1,2,snd_value);
  }
//+------------------------------------------------------------------+
//| 通过空值填充指标缓冲区的i元素                                    |
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
  {
//--- 在指标缓冲区单元设置空值
   ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayMinimum

搜索多维度数值数组的第一维度中的最低元素。

```
int  ArrayMinimum(
   const void&   array[],              // 用来搜索的数组
   int           start=0,              // 启动用于检测的指标
   int           count=WHOLE_ARRAY     // 检测元素数量
   );
```

参量

*array[]*

[in] 搜索数字数组

*start=0*

[in] 标引开始检测

*count=WHOLE_ARRAY*

[in] 搜索元素数量，默认搜索全部数组 (count=WHOLE_ARRAY).

返回值

函数返回发现元素的标引索引，考虑数组系列，失败返回-1

注释

搜索最小值时考虑 AS_SERIES 标识值。

函数ArrayMaximum 和 ArrayMinimum 接受任何维度的数组为参数。然而，搜索却始终应用于第一(0)维度。

示例：

```
#property description "The indicator displays larger time frame's candlesticks on the
//--- 指标设置
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- 图 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- 图 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- 图 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- 图 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
//---- 图 5
#property indicator_label5  "BullBody"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color5  clrOlive,clrOlive
//---- 图 6
#property indicator_label6  "BullBodyEnd"
#property indicator_color6  clrOlive,clrOlive
//---- 图 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- 图 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- 预定义常量
#define INDICATOR_EMPTY_VALUE 0.0
//--- 输入参数
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // 指标计算的时间范围
input datetime        InpDateStart=D'2013.01.01 00:00'; // 分析起始日期
//--- 熊市蜡烛图的指标缓冲区
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- 牛市蜡烛图的指标缓冲区
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- 全局变量
datetime ExtTimeBuff[];      // 较大时间帧的时间缓冲区
int      ExtSize=0;          // 时间缓冲区大小
int      ExtCount=0;         // 时间缓冲区的标引
int      ExtStartPos=0;      // 指标计算的初始位置
bool     ExtStartFlag=true;  // 接收初始位置的辅助标识
datetime ExtCurrentTime[1];  // 生成较大时间帧柱形的最近时间
datetime ExtLastTime;        // 较大时间帧的最近时间，执行计算
bool     ExtBearFlag=true;   // 标记定义编写数据订单熊市指标缓冲区
bool     ExtBullFlag=true;   // 标记定义编写数据订单牛市指标缓冲区
int      ExtIndexMax=0;      // 数组中最大元素标引
int      ExtIndexMin=0;      // 数组中最小元素标引
int      ExtDirectionFlag=0; // 当前蜡烛图的价格移动方向
//--- 正确绘制蜡烛图开盘价和收盘价之间的转换
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 填写蜡烛图的基本部分                                                |
//+------------------------------------------------------------------+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- 在数组中找出最大和最小元素的标引
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // 最高极值
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // 最低极值
//--- 定义当前时间帧填充多少柱
   int count=fill_index-start+1;
//--- 如果首柱的收盘价超过最后的柱，那么蜡烛图为熊市
   if(open[start]>close[last])
     {
      //--- 如果蜡烛图之前牛市，清空牛市指标缓冲区的值
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
      //--- 熊市蜡烛图
      ExtDirectionFlag=-1;
      //--- 生成蜡烛图
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     close[last],high[index_max],low[index_min],start,count,ExtBearFla
      //--- 退出函数
      return;
     }
//--- 如果首柱的收盘价少于最后的柱，那么蜡烛图为牛市
   if(open[start]<close[last])
     {
      //--- 如果蜡烛图之前为熊市，清空熊市指标缓冲区的值
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
      //--- 牛市蜡烛图
      ExtDirectionFlag=1;
      //--- 生成蜡烛图
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
                     open[start],high[index_max],low[index_min],start,count,ExtBullFla
      //--- 退出函数
      return;
     }
//--- 如果您在这部分函数，首柱开盘价等于
//--- 最后柱的收盘价;这种蜡烛图被认为熊市图
//--- 如果在那之前蜡烛图为牛市，清空牛市指标缓冲区的值
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowS
//--- 熊市蜡烛图
   ExtDirectionFlag=-1;
//--- 如果收盘和开盘价相同，使用切换来正确显示
   if(high[index_max]!=low[index_min])
```

© 2000-2025, MetaQuotes Ltd.

```
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start],open[start]-ExtEmptyBodySize,high[index_max],
                     high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| 填充蜡烛图末端                                                   |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- 单柱时不要绘制
   if(last-start==0)
      return;
//--- 如果首柱的收盘价超过最后的柱，蜡烛图为熊市
   if(open[start]>close[last])
     {
      //--- 生成蜡烛图末端
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start],close[last],high[index_max],low[index_min],fill_index,
      //--- 退出函数
      return;
     }
//--- 如果首柱的收盘价小于最后的柱，蜡烛图为牛市
   if(open[start]<close[last])
     {
      //--- 生成蜡烛图末端
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ext
                    close[last],open[start],high[index_max],low[index_min],fill_index,
      //--- 退出函数
      return;
     }
//--- 如果您在这部分函数，首柱开盘价等于
//--- 最后柱的收盘价;这种蜡烛图被认为熊市图
//--- 生成蜡烛图末端
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_i
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEm
  }
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int OnInit()
  {
//--- 检查指标周期
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- 在前景显示价格数据
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- 绑定指标缓冲区
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- 设置一些属性值来创建指标
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // 图形结构类型
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // 绘制线型风格
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // 绘制线型宽度
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 如果没有计算的柱
```

© 2000-2025, MetaQuotes Ltd.

```
   if(prev_calculated==0)
     {
      //--- 接收较大时间帧柱形的到达时间
      if(!GetTimeData())
         return(0);
     }
//--- 设置直接索引
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- 启动计算柱形的变量
   int start=prev_calculated;
//--- 如果生成柱形，重新计算上面的指标值
   if(start!=0 && start==rates_total)
      start--;
//--- 循环计算指标值
   for(int i=start;i<rates_total;i++)
     {
      //--- 用空值填充指标缓冲区的i元素
      FillIndicatorBuffers(i);
      //--- 从InpDateStart日期开始计算柱形
      if(time[i]>=InpDateStart)
        {
         //--- 定义值第一时间显示的持仓
         if(ExtStartFlag)
           {
            //--- 存储最初柱形的数量
            ExtStartPos=i;
            //--- 定义超过time[i]的较大时间帧的初始日期
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- 改变标识值以便不在这模块再次运行
            ExtStartFlag=false;
           }
         //--- 检查数组中是否有任何元素
         if(ExtCount<ExtSize)
           {
            //--- 等候当前时间帧的值到达较大时间帧的值
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //---  绘制蜡烛图的主要部分（不填写最后和倒数第二柱之间的区域）
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
               //--- 填充蜡烛图末端（最后和倒数第二柱之间的区域）
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
               //--- 切换初始持仓以绘制下一个蜡烛图
               ExtStartPos=i;
```

© 2000-2025, MetaQuotes Ltd.

```
                //--- 增加数组计数器
                ExtCount++;
               }
             else
                continue;
            }
          else
            {
             //--- 重置数组的值
             ResetLastError();
             //--- 接收来自较大时间帧的最后日期
             if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
               {
                Print("Data copy error, code = ",GetLastError());
                return(0);
               }
             //--- 如果新日期靠后，停止生成蜡烛图
             if(ExtCurrentTime[0]>ExtLastTime)
               {
                //--- 在主要指标缓冲区清空最后和倒数第二柱之间的区域
                ClearEndOfBodyMain(i-1);
                //--- 使用辅助指标缓冲区填充该区域
                FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
                //--- 切换初始持仓以绘制下一个蜡烛图
                ExtStartPos=i;
                //--- 重置价格方向标识
                ExtDirectionFlag=0;
                //--- 存储新的最近日期
                ExtLastTime=ExtCurrentTime[0];
               }
             else
               {
                //--- 生成蜡烛图
                FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndex
               }
            }
         }
      }
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 检查指定指标周期的正确性                                         |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
//--- 指标周期应该超过显示它的时间帧
   if(current_period>=high_period)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("Error! The value of the indicator period should exceed the value of the c
      return(false);
     }
//--- 如果指标周期是周或月份，周期则是正确的
   if(high_period>32768)
      return(true);
//--- 转变周期值到分钟
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- 指标周期应该是它显示的时间帧的几倍
   if(high_period%current_period!=0)
     {
      Print("Error! The value of the indicator period should be multiple of the value
      return(false);
     }
//--- 指标周期应该超过它显示的时间帧的3或更多倍
   if(high_period/current_period<3)
     {
      Print("Error! The indicator period should exceed the current time frame 3 or mo
      return(false);
     }
//--- 指标周期在当前时间帧是正确的
   return(true);
  }
//+------------------------------------------------------------------+
//| 接收来自较大时间帧的时间数据                                     |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- 重置错误的值
   ResetLastError();
//--- 复制当前时间的所有数据
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- 接收错误代码
      int code=GetLastError();
      //--- 打印错误信息
      PrintFormat("Data copy error! %s",code==4401
                  ? "History is still being uploaded!"
                  : "Code = "+IntegerToString(code));
      //--- 返回false，反复尝试下载数据
      return(false);
     }
//--- 接收数组大小
   ExtSize=ArraySize(ExtTimeBuff);
//--- 为数组设置循环指数为零
   ExtCount=0;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在时间帧设置当前蜡烛图位置为零
   ExtStartPos=0;
   ExtStartFlag=true;
//--- 存储来自较大时间帧的最近时间值
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- 成功执行
   return(true);
  }
//+------------------------------------------------------------------------------+
//| 函数形成了蜡烛图的主要部分。根据标识的                                          |
//| 值，函数定义了要使用哪个数据和数组                                              |
//| 以便正确显示                                                                  |
//+------------------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- 检查标识的值
   if(flag)
     {
      //--- 生成蜡烛图主体
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- 生成蜡烛图影子
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- 生成蜡烛图主体
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- 生成蜡烛图影子
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+------------------------------------------------------------------------------+
//| 函数形成了蜡烛图的末端。根据标识的值，                                          |
//| 函数定义了要使用哪个数据和数组                                                  |
//| 以便正确显示。                                                                |
//+------------------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
                   const int end,bool &flag)
  {
//--- 检查标识的值
   if(flag)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 生成蜡烛图主体的末端
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- 生成蜡烛图影子的末端
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- 改变标识值到相反方向
      flag=false;
     }
   else
     {
      //--- 生成蜡烛图主体的末端
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- 生成蜡烛图影子的末端
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- 改变标识值到相反方向
      flag=true;
     }
  }
//+------------------------------------------------------------------------------------
//| 清空蜡烛图末端 (最后和倒数第二柱之间的区域)
//|
//+------------------------------------------------------------------------------------
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecc
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecc
  }
//+------------------------------------------------------------------+
//| 清空蜡烛图                                                       |
//+------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- 检查
   if(count!=0)
     {
      //--- 以空值填充指标缓冲区
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+------------------------------------------------------------------+
//| 生成蜡烛图的主要部分                                             |
//+------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- 检查
```

© 2000-2025, MetaQuotes Ltd.

```
    if(count!=0)
      {
       //--- 用值填充指标缓冲区
       ArrayFill(fst,start,count,fst_value);
       ArrayFill(snd,start,count,snd_value);
      }
   }
//+------------------------------------------------------------------+
//| 生成蜡烛图末端                                                   |
//+------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
   {
//--- 用值填充指标缓冲区
    ArrayFill(fst,last-1,2,fst_value);
    ArrayFill(snd,last-1,2,snd_value);
   }
//+------------------------------------------------------------------+
//| 通过空值填充指标缓冲区的i元素                                    |
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
   {
//--- 在指标缓冲区单元设置空值
    ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayPrint

将简单类型或简单结构的数组打印到日志。

```
void  ArrayPrint(
   const void&   array[],              // 打印数组
   uint          digits=_Digits,       // 小数位数
   const string  separator=NULL,       // 结构字段值的分隔符
   ulong         start=0,              // 最初打印元素索引
   ulong         count=WHOLE_ARRAY,    // 打印元素数量
   ulong         flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_
   );
```

参数

*array[]*

[in] 简单类型或简单结构的数组。

*digits=_Digits*

[in] 真实类型的小数位数。默认值是_Digits。

*separator=NULL*

[in] 结构元素字段值的分隔符。NULL的默认值为空行。这种情况下，使用空格作为分隔符。

*start=0*

[in] 最初打印数组元素的索引。索引默认从0开始。

*count=WHOLE_ARRAY*

[in] 将要打印的数组元素数。整个数组默认显示(count=WHOLE_ARRAY)。

*flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_ALIGN*

[in] 标识组合设置输出模式。全部标识默认启用：

- ARRAYPRINT_HEADER -打印结构数组标题
- ARRAYPRINT_INDEX -在左侧打印索引
- ARRAYPRINT_LIMIT - 仅打印最初100个和最后100个数组元素。只在您只想打印大型数组一部分的时候使用。
- ARRAYPRINT_ALIGN -启用对齐打印值 -数字右侧对齐，而行是左侧对齐。
- ARRAYPRINT_DATE -打印datetime时，以dd.mm.yyyy格式打印日期
- ARRAYPRINT_MINUTES -打印datetime时，以HH:MM格式打印时间
- ARRAYPRINT_SECONDS -打印datetime时，以HH:MM:SS格式打印时间

返回值

No

注意

ArrayPrint() 不会将全部结构数组字段打印到日志 -跳过数组和对象指针字段。之所以没有打印这些列只是为了更方便的展示。如果您需要打印全部结构字段，您需要按照所需的格式编写您自己的print函数。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//--- 打印最后10 个柱形图的值
   MqlRates rates[];
   if(CopyRates(_Symbol,_Period,1,10,rates))
     {
      ArrayPrint(rates);
      Print("Check\n[time]\t[open]\t[high]\t[low]\t[close]\t[tick_volume]\t[spread]\t
      for(int i=0;i<10;i++)
        {
         PrintFormat("[%d]\t%s\t%G\t%G\t%G\t%G\t%G\t%G\t%I64d\t",i,
         TimeToString(rates[i].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS),
         rates[i].open,rates[i].high,rates[i].low,rates[i].close,
         rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
     }
   else
      PrintFormat("CopyRates failed, error code=%d",GetLastError());
//--- 打印示例
/*
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295         18110       10   17
   [1] 2016.11.09 05:00:00 1.12296 1.12825 1.11930 1.12747         17829        9   15
   [2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744         13458       10    9
   [3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194         15362        9   12
   [4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172         16833        9   12
   [5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052         15933        8   10
   [6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528         11888        9    8
   [7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915          7284       10    5
   [8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904          8710        9    6
   [9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263          8956        7    7
  检查
   [time] [open] [high] [low] [close] [tick_volume] [spread] [real_volume]
   [0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295 18110 10 17300175000
   [1] 2016.11.09 05:00:00 1.12296 1.12825 1.1193 1.12747 17829 9 15632176000
   [2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744 13458 10 9593492000
   [3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194 15362 9 12352245000
   [4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172 16833 9 12961333000
   [5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052 15933 8 10720384000
   [6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528 11888 9 8084811000
   [7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915 7284 10 5087113000
   [8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904 8710 9 6769629000
   [9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263 8956 7 7192138000
*/
```

另 见

FileSave, FileLoad

© 2000-2025, MetaQuotes Ltd.

# ArrayRange

在所选数组规格中该函数返回元素数字

```
int  ArrayRange(
   const void&   array[],      // 用于检测的数组
   int           rank_index    // 维数
   );
```

参量

*array[]*

[in] 检测数组

*rank_index*

[in] 标准索引

返回值

在所选数组规格中的元素数量

注释

由于标引始于零，所以阵列维度的数量比上一个维度标引大一个。

示例：

```
void OnStart()
  {
//--- 创建四维数组
   double array[][5][2][4];
//--- 设置零维度大小
   ArrayResize(array,10,10);
//--- 打印维度
   int temp;
   for(int i=0;i<4;i++)
     {
      //--- 接收i维度大小
      temp=ArrayRange(array,i);
      //--- 打印
      PrintFormat("dim = %d, range = %d",i,temp);
     }
//--- 结果
// dim = 0, range = 10
// dim = 1, range = 5
// dim = 2, range = 2
// dim = 3, range = 4
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrayResize

函数建立新的第一维大小

```
int  ArrayResize(
   void&  array[],              // 引用传递的数组
   int    new_size,             // 新数组大小
   int    reserve_size=0        // 保留尺寸值（过量）
   );
```

参量

*array[]*

[out] 数组变化尺寸

*new_size*

[in] 第一维度新尺寸

*reserve_size=0*

[in] 获得存储的分散式大小

返回值

如果执行成功，在数组中改变规格后返回所有元素计数，否则，返回-1，数组不调整大小。

如果ArrayResize() 应用于静态数组，timeseries或指标缓冲区，数组大小保持一致 这些数组将不会被重新分配。在这种情况下，如果*new_size*<=ArraySize(*数组*)，函数将只会返回 *new_size*；否则将会返回-1值。

注释

该函数只能应用在 动态数组。应该注意的是您不能改变通过SetIndexBuffer()函数分配的作为指标缓冲区的动态数组的大小。对于指标缓冲区，调整大小的所有操作都通过程序端的运行时间子系统来执行。

数组中的元素总数不能超过 2147483647。

随着频繁的内存分配，推荐使用设置存储以减少物理内存分配数量的第三方参数。后来的ArrayResize所有调用都不能导致重新分配物理内存，只是会在保留内存内改变第一数组维度的大小。应谨记，第三方参数只能在物理内存分配期间使用。例如：

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
   ArrayResize(arr,i,1000);
```

假若这样，内存将被重新分配两次，第一次是输入2000元素循环之前 数组大小将设为1000)，第二次是i等于3000.如果我们跳过第三方参数，将会有2000重新分配物理内存，这将使程序减速。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 计数器
   ulong start=GetTickCount();
   ulong now;
   int   count=0;
//--- 用于快速版本维度的数组
   double arr[];
   ArrayResize(arr,100000,100000);
//--- 检查如何加速内存保留工作的变量
   Print("--- Test Fast: ArrayResize(arr,100000,100000)");
   for(int i=1;i<=300000;i++)
     {
      //--- 设置新数组大小指定储备100,000元素!
      ArrayResize(arr,i,100000);
      //--- 当到达整十整百的整数时,显示数组大小和花费的时间
      if(ArraySize(arr)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(arr),(now-sta
         start=now;
        }
     }
//--- 现在显示,无保留内存的版本有多慢
   double slow[];
   ArrayResize(slow,100000,100000);
//---
   count=0;
   start=GetTickCount();
   Print("---- Test Slow: ArrayResize(slow,100000)");
//---
   for(int i=1;i<=300000;i++)
     {
      //--- 设置新的数组大小,但是无额外储备
      ArrayResize(slow,i);
      //--- 当到达整十整百的整数时,显示数组大小和花费的时间
      if(ArraySize(slow)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(slow)=%d Time=%d ms",count,ArraySize(slow),(now-st
         start=now;
        }
     }
  }
//--- 脚本的示例结果
/*
   Test_ArrayResize (EURUSD,H1)   --- Test Fast: ArrayResize(arr,100000,100000)
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(arr)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(arr)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(arr)=300000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   ---- Test Slow: ArrayResize(slow,100000)
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(slow)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(slow)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(slow)=300000 Time=228511 ms
*/
```

© 2000-2025, MetaQuotes Ltd.

另 见

ArrayInitialize

© 2000-2025, MetaQuotes Ltd.

# ArrayInsert

从指定索引开始，将来自源数组的指定元素数量插入到接收数组。

```
bool  ArrayInsert(
   void&         dst_array[],          // 接收数组
   const void&   src_array[],          // 源数组
   uint          dst_start,            // 即将插入的接收方数组索引
   uint          src_start=0,          // 即将复制的源数组索引
   uint          count=WHOLE_ARRAY     // 即将插入的元素数量
   );
```

参数

*dst_array[]*

[in][out] 应该添加的元素的接收数组。

*src_array[]*

[in] 即将添加的元素的源数组。

*dst_start*

[in] 从源数组插入元素的接收数组中的索引。

*src_start=0*

[in] 接收数组的索引，从源数组插入元素。

*count*

[in] 从源数组中添加的元素的数量。WHOLE_ARRAY表示从指定索引到数组结束的所有元素。

返回值

如果成功返回true，否则返回false。若要获得错误信息，请调用GetLastError()函数。可能错误：

- 5052 - ERR_SMALL_ARRAY（*start*和/或*count*参数被错误设置或*src_array[]*源数组为空），
- 5056 - ERR_SERIES_ARRAY（该数组不能更改，指标缓冲区），
- 4006 - ERR_INVALID_ARRAY（不允许自复制，或数组类型不同，或具有包含类对象或析构结构的固定大小数组），
- 4005 - ERR_STRUCT_WITHOBJECTS_ORCLASS（数组不包含POD结构，表示不可能进行简单复制），
- 当在ArrayRemove()函数描述中提供了更改*dst_array[]*接收数组大小时发生的错误。

注意

如果函数用于固定大小的数组，则*dst_array[]*接收数组本身的大小不变。从*dst_start*位置开始，接收数组的元素向右移动（最后一个*counts*元素"脱离"），而从源数组复制的元素则取而代之。

您不能将元素插入到由SetIndexBuffer()函数指定为指标缓冲区的动态数组中。对于指标缓冲区，所有更改大小的操作都通过程序端的执行子系统来完成。

© 2000-2025, MetaQuotes Ltd.

在源数组中，元素从*src_start*索引开始复制。源数组大小保持不变。将要添加到接收数组中的元素不会链接到源数组元素中。这意味着任何两个数组中元素的后续更改都不会反映在第二个数组中。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明固定大小数组并填充值
   int array_dest[10];
   for(int i=0;i<10;i++)
     {
      array_dest[i]=i;
     }
   //--- 源数组
   int array_source[10];
   for(int i=0;i<10;i++)
     {
      array_source[i]=10+i;
     }
//--- 在插入元素之前显示数组
   Print("Before calling ArrayInsert()");
   ArrayPrint(array_dest);
   ArrayPrint(array_source);
//--- 插入源数组的3个元素，显示接收数组的新集合
   ArrayInsert(array_dest,array_source,4,0,3);
   Print("After calling ArrayInsert()");
   ArrayPrint(array_dest);
/*
  Execution result
   Before calling ArrayInsert()
   0 1 2 3 4 5 6 7 8 9
   After calling ArrayInsert()
   0 1 2 3 10 11 12 7 8 9
*/
```

另见

ArrayRemove, ArrayCopy, ArrayResize, ArrayFree

© 2000-2025, MetaQuotes Ltd.

# ArrayRemove

从指定索引开始移除数组中指定的元素数量。

```
bool  ArrayRemove(
   void&         array[],             // 任何类型的数组
   uint          start,               // 开始移除的索引
   uint          count=WHOLE_ARRAY    // 元素数量
   );
```

参数

*array[]*

[in][out] 数组。

*start*

[in] 索引，数组元素从这里开始移除。

*count=WHOLE_ARRAY*

[in] 移除元素的数量。WHOLE_ARRAY值表示移除从指定索引到数组结束的所有元素。

返回值

如果成功返回true，否则返回false。若要获得错误信息，请调用GetLastError()函数。可能错误：

- 5052 - ERR_SMALL_ARRAY(*start*值过大)，
- 5056 -ERR_SERIES_ARRAY (该数组不能更改，指标缓冲区)，
- 4003 - ERR_INVALID_PARAMETER(*count*值过大)，
- 4005 - ERR_STRUCT_WITHOBJECTS_ORCLASS (包含具有析构函数的复杂对象的固定大小数组)，
- 4006 -ERR_INVALID_ARRAY (包含具有析构函数的结构或类对象的固定大小数组)。

注意

如果函数用于固定大小的数组，该数组大小不变：剩余的"tail"物理复制到*start*位置。对于准确理解函数如何工作，请参阅以下示例。"Physical(物理)"复制表示复制的对象不是通过调用构造函数或复制操作符创建的。而是复制对象的二进制表示法。为此，您不可以将ArrayRemove()函数应用于包含析构函数对象的固定大小数组 (激活ERR_INVALID_ARRAY或ERR_STRUCT_WITHOBJECTS_ORCLASS错误)。当在移除这种对象时，应该调用两次析构函数 - 针对原始对象及其副本。

您不能从由SetIndexBuffer()函数指定为指标缓冲区的动态数组移除元素。这会导致ERR_SERIES_ARRAY错误。对于指标缓冲区，所有更改大小的操作都通过程序端的执行子系统来完成。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明固定大小数组并填充值
```

© 2000-2025, MetaQuotes Ltd.

```
   int array[10];
   for(int i=0;i<10;i++)
     {
      array[i]=i;
     }
//--- 在移除元素之前显示数组
   Print("Before calling ArrayRemove()");
   ArrayPrint(array);
//--- 从数组中删除2个元素并显示新集合
   ArrayRemove(array,4,2);
   Print("After calling ArrayRemove()");
   ArrayPrint(array);
/*
  执行结果:
  Before calling ArrayRemove()
   0 1 2 3 4 5 6 7 8 9
  After calling ArrayRemove()
   0 1 2 3 6 7 8 9 8 9
*/
```

另 见

ArrayInsert, ArrayCopy, ArrayResize, ArrayFree

© 2000-2025, MetaQuotes Ltd.

# ArrayReverse

从指定索引开始反转数组中指定的元素数量。

```
bool  ArrayReverse(
   void&         array[],            // 任何类型的数组
   uint          start=0,            // 开始反转数组的索引
   uint          count=WHOLE_ARRAY   // 元素数量
   );
```

参数

*array[]*

[in][out] 数组。

*start=0*

[in] 开始反转的数组的索引。

*count=WHOLE_ARRAY*

[in] 反转的元素数量。如果WHOLE_ARRAY，那么，从指定的*start*索引开始以倒序方式移动索引数组元素，直到数组结束。 index up to the end of the array.

返回值

如果成功返回true，否则返回false。

注意

ArraySetAsSeries()函数不会物理移动数组元素。相反，它只反向更改索引方向来安排对元素的访问，如同在timeseries中一样。ArrayReverse()函数可以物理移动数组元素，使数组“反转”。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明固定大小数组并填充值
   int array[10];
   for(int i=0;i<10;i++)
     {
      array[i]=i;
     }
//--- 在反转元素之前显示数组
   Print("Before calling ArrayReverse()");
   ArrayPrint(array);
//--- 反转数组中的3个元素并显示新集合
   ArrayReverse(array,4,3);
   Print("After calling ArrayReverse()");
   ArrayPrint(array);
/*
```

© 2000-2025, MetaQuotes Ltd.

```
   执行结果:
   Before calling ArrayReverse()
    0 1 2 3 4 5 6 7 8 9
   After calling ArrayReverse()
    0 1 2 3 6 5 4 7 8 9
*/
```

另见

ArrayInsert, ArrayRemove, ArrayCopy, ArrayResize, ArrayFree, ArrayGetAsSeries, ArraySetAsSeries

© 2000-2025, MetaQuotes Ltd.

# ArraySetAsSeries

设置标记到选定的动态数组对象的函数，其元件如时间序列中一样被索引。

```
bool  ArraySetAsSeries(
   const void&  array[],     // 通过引用的数组
   bool         flag         // true表示倒序索引
   );
```

参量

*array[]*

[in][out] 设置数字数组

*flag*

[in] 数组索引方向

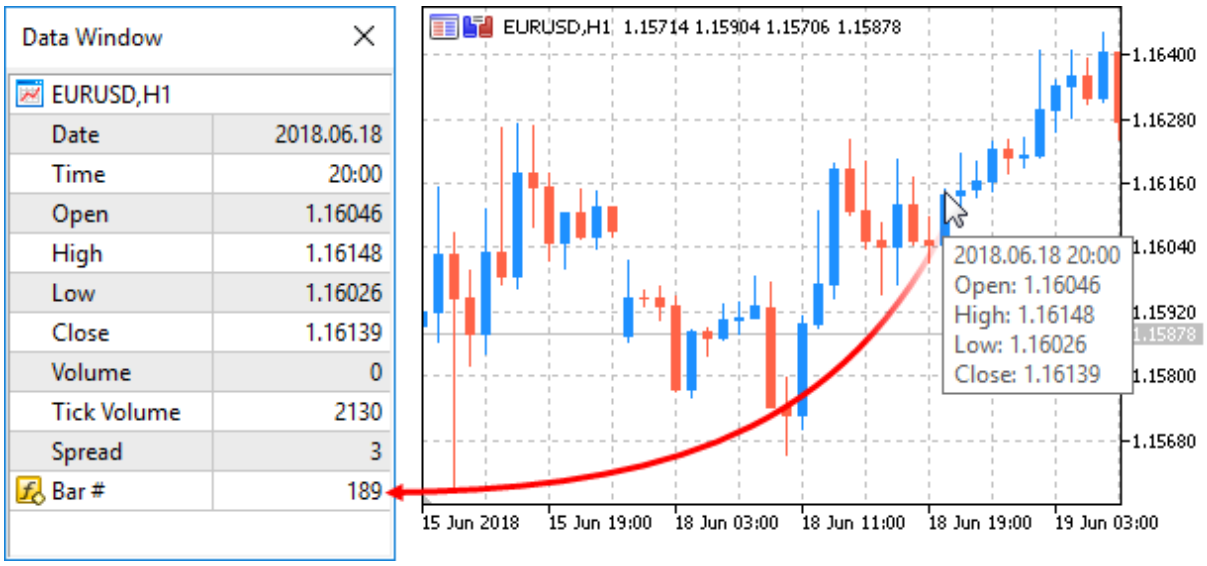
返回值

成功函数返回true，否则 - false.

注释

不能为多维数组或静态数组 (数组，方框中的大小在编译步骤已经存在)设置 AS_SERIES 标记。 时间序列中的索引不同于时间序列元件中普通数组，从后往前 (从最新的到最原始数据)。

示例：显示柱值的指标




© 2000-2025, MetaQuotes Ltd.

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图的编号
#property indicator_label1  "Numeration"
#property indicator_type1   DRAW_LINE
#property indicator_color1  CLR_NONE
//--- 指标缓冲区
double         NumerationBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 自定义缓冲区绘图
   SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- 为类似时间序列的缓冲区设置指标
   ArraySetAsSeries(NumerationBuffer,true);
//--- 设置数据窗口中显示的精确度
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- 标识符数组名称在数据窗口中如何显示
   PlotIndexSetString(0,PLOT_LABEL,"Bar #");
//---
   return(0);
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---  存储当前打开的零柱时间
   static datetime currentBarTimeOpen=0;
//--- 返回访问数组时间[] - 像时间序列一样
   ArraySetAsSeries(time,true);
//--- 如果零柱时间不同于已经存储的
   if(currentBarTimeOpen!=time[0])
     {
     //--- 从当前图表到图表深度列举所有柱
      for(int i=rates_total-1;i>=0;i--) NumerationBuffer[i]=i;
      currentBarTimeOpen=time[0];
     }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

另见

接入时间序列，ArrayGetAsSeries

© 2000-2025, MetaQuotes Ltd.

# ArraySize

函数返回所选数组的元素数量

```
int  ArraySize(
   const void&  array[]    // 检测的数组
   );
```

参量

*array[]*

[in] 任意类型的数组

返回值

整型 值

注释

一维数组，值可以通过 ArraySize 返回，等于 ArrayRange(数组,0)

示例：

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
//--- 创建数组
   double one_dim[];
   double four_dim[][10][5][2];
//--- 大小
   int one_dim_size=25;
   int reserve=20;
   int four_dim_size=5;
//--- 辅助变量
   int size;
//--- 分配内存没有备份
   ArrayResize(one_dim,one_dim_size);
   ArrayResize(four_dim,four_dim_size);
//--- 1. 一维数组
   Print("+============================================================+");
   Print("Array sizes:");
   Print("1. One-dimensional array");
   size=ArraySize(one_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",one_dim_size,size);
//--- 2. 多维数组
   Print("2. Multidimensional array");
   size=ArraySize(four_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",four_dim_size,size);
//--- 维度大小
   int d_1=ArrayRange(four_dim,1);
   int d_2=ArrayRange(four_dim,2);
   int d_3=ArrayRange(four_dim,3);
   Print("Check:");
   Print("Zero dimension = Array size / (First dimension * Second dimension * Third di
   PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,d_3);
//--- 3. 备份内存的一维数组
   Print("3. One-dimensional array with memory backup");
//--- 双倍的值
   one_dim_size*=2;
//--- 分配备份内存
   ArrayResize(one_dim,one_dim_size,reserve);
//--- 打印大小
   size=ArraySize(one_dim);
   PrintFormat("Size with backup = %d, Actual array size = %d",one_dim_size+reserve,si
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArraySort

按升序排列排序多维度数值数组的第一维度中的值。

```
bool  ArraySort(
   void&  array[]      // 数组排序
   );
```

参量

*array[]*

[in][out] 数字数组分类

返回值

成功返回真值，否则是错误值。

注释

数组通过 AS_SERIES 标签建立，并降序排列。

注释

无论AS_SERIES标识是何值，始终按照升序排列分类数组。

函数ArraySort 和 ArrayBSearch 接受任何维度的数组为参数。然而，搜索和分类却始终应用于第一(0)维度。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property description "The indicator analyzes data for the last month and draws all ca
#property description "and large tick volumes. The tick volume array is sorted out"
#property description "to define such candlesticks. The candlesticks having the volume
#property description "per cent of the array are considered small. The candlesticks ha
#property description "the last InpBigVolume per cent of the array are considered larg
//--- 指标设置
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//--- 图
#property indicator_label1  "VolumeFactor"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrDodgerBlue,clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- 预定义常量
#define INDICATOR_EMPTY_VALUE 0.0
//--- 输入参数
input int InpSmallVolume=15; // 小型交易量的百分比值 (<50)
input int InpBigVolume=20;   // 大型交易量的百分比值 (<50)
//--- 分析起始时间 (将会改变)
datetime ExtStartTime;
//--- 指标缓存区
double   ExtOpenBuff[];
double   ExtHighBuff[];
double   ExtLowBuff[];
double   ExtCloseBuff[];
double   ExtColorBuff[];
//--- 显示蜡烛图的交易量边界值
long     ExtLeftBorder=0;
long     ExtRightBorder=0;
//+------------------------------------------------------------------+
//| 接收跳动量的边界值                                               |
//+------------------------------------------------------------------+
bool GetVolumeBorders(void)
  {
//--- 变量
   datetime stop_time;  // 复制结束时间
   long     buff[];     // 用于复制的缓冲区
//--- 结束时间是当前时间
   stop_time=TimeCurrent();
//--- 起始时间比当前时间早一个月
   ExtStartTime=GetStartTime(stop_time);
//--- 接收跳动量的值
   ResetLastError();
   if(CopyTickVolume(Symbol(),Period(),ExtStartTime,stop_time,buff)==-1)
     {
      //--- 接收数据失败，返回false启动重新计算的命令
      PrintFormat("Failed to receive tick volume values. Error code = %d",GetLastError
      return(false);
     }
//--- 计算数组大小
   int size=ArraySize(buff);
//--- 分类数组
   ArraySort(buff);
//--- 定义跳动量左侧和右侧的边界值
   ExtLeftBorder=buff[size*InpSmallVolume/100];
   ExtRightBorder=buff[(size-1)*(100-InpBigVolume)/100];
//--- 成功执行
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 接收比已过月份少一个月的数据                                     |
//+------------------------------------------------------------------+
datetime GetStartTime(const datetime stop_time)
  {
//--- 转换结束时间到MqlDateTime类型结构变量
   MqlDateTime temp;
   TimeToStruct(stop_time,temp);
//--- 接收少一个月的数据
   if(temp.mon>1)
      temp.mon-=1;  // 当前月份不是今年的第一个月，因此之前月份减一
   else
     {
      temp.mon=12;  // 当前月份是今年的第一个月，因此，前一个月是12月，
      temp.year-=1; // 而年份数减一
     }
//--- 天数将不会超过28
   if(temp.day>28)
      temp.day=28;
//--- 返回获得的日期
   return(StructToTime(temp));
  }
//+------------------------------------------------------------------+
//| 自定义指标初始函数                                               |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 检查是否输入参数满足条件
   if(InpSmallVolume<0 || InpSmallVolume>=50 || InpBigVolume<0 || InpBigVolume>=50)
     {
      Print("Incorrect input parameters");
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- 指标缓冲区映射
   SetIndexBuffer(0,ExtOpenBuff);
   SetIndexBuffer(1,ExtHighBuff);
   SetIndexBuffer(2,ExtLowBuff);
   SetIndexBuffer(3,ExtCloseBuff);
   SetIndexBuffer(4,ExtColorBuff,INDICATOR_COLOR_INDEX);
//--- 设置不被显示的值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- 设置指标缓冲区标签
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 检查未处理的柱形是否仍然存在
```

© 2000-2025, MetaQuotes Ltd.

```
   if(prev_calculated<rates_total)
     {
      //--- 接收交易量右侧和左侧的新边界值
      if(!GetVolumeBorders())
         return(0);
     }
//--- 启动柱形计算的变量
   int start=prev_calculated;
//--- 如果在前一个订单号已经计算指标值，那么使用最近的柱形
   if(start>0)
      start--;
//--- 设置时间序列的直接标引
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
   ArraySetAsSeries(tick_volume,false);
//--- 计算指标值的循环
   for(int i=start;i<rates_total;i++)
     {
      //--- 从初始日期填写蜡烛图
      if(ExtStartTime<=time[i])
        {
         //--- 如果值不少于右侧边界，则填写蜡烛图
         if(tick_volume[i]>=ExtRightBorder)
           {
            //--- 接收数据绘制蜡烛图
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- 闪蓝色
            ExtColorBuff[i]=0;
            //--- 继续循环
            continue;
           }
         //--- 如果值不超过左侧边界，则填写蜡烛图
         if(tick_volume[i]<=ExtLeftBorder)
           {
            //--- 接收数据绘制蜡烛图
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- 桔色
            ExtColorBuff[i]=1;
            //--- 继续循环
            continue;
           }
        }
      //--- 为计算中不包含的柱设置空值
      ExtOpenBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtHighBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtLowBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtCloseBuff[i]=INDICATOR_EMPTY_VALUE;
     }
//--- 返回prev_calculated的值以便下次调用
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

另 见

ArrayBsearch

© 2000-2025, MetaQuotes Ltd.

# ArraySwap

交换相同类型两个动态数组的内容。对于多维数组而言，除了第一个之外，所有维度中的元素数量都应该匹配。

```
bool  ArraySwap(
   void&  array1[],      //第一个数组
   void&  array2[]       //第二个数组
   );
```

参数

*array1[]*

[in][out] 数值类型的数组。

*array2[]*

[in][out] 数值类型的数组。

返回值

如果成功返回true，否则返回false。在这种情况下，GetLastError()返回ERR_INVALID_ARRAY错误代码。

注意

该函数会接受相同类型的动态数组和除了第一个维度外的相同维度的数组。对于整数类型，可以忽略符号，例如char==uchar)

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 存储报价的数组
   double source_array[][8];
   double   dest_array[][8];
   MqlRates rates[];
//--- 获得当前时间周期最新20个蜡烛图的数据
   int copied=CopyRates(NULL,0,0,20,rates);
   if(copied<=0)
     {
      PrintFormat("CopyRates(%s,0,0,20,rates) failed, error=%d",
                  Symbol(),GetLastError());
      return;
     }
//--- 为已复制的数据数量设置数组大小
   ArrayResize(source_array,copied);
//--- 使用来自rates[]的数据填写rate_array_1[]数组
   for(int i=0;i<copied;i++)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      source_array[i][0]=(double)rates[i].time;
      source_array[i][1]=rates[i].open;
      source_array[i][2]=rates[i].high;
      source_array[i][3]=rates[i].low;
      source_array[i][4]=rates[i].close;
      source_array[i][5]=(double)rates[i].tick_volume;
      source_array[i][6]=(double)rates[i].spread;
      source_array[i][7]=(double)rates[i].real_volume;
     }
//--- 交换source_array[] 和dest_array[]之间的数据
   if(!ArraySwap(source_array,dest_array))
     {
      PrintFormat("ArraySwap(source_array,rate_array_2) failed, error code=%d",GetLast
      return;
     }
//--- 确保原数组在交换之后变为零
   PrintFormat("ArraySwap() done: ArraySize(source_array)=%d",ArraySize(source_array))
//--- 展示dest_array[]目标数组的数据
   ArrayPrint(dest_array);
  }
```

另 见

ArrayCopy, ArrayFill, ArrayRange, ArrayIsDynamic

© 2000-2025, MetaQuotes Ltd.

# ArrayToFP16

将float或double类型数组复制到给定格式的ushort类型的数组中。

```
bool   ArrayToFP16(
   const ushort&         dst_array[],         // 复制到
   const float&          src_array[],         // 复制自
   ENUM_FLOAT16_FORMAT   fmt                  // 格式
   );
```

double类型的重载

```
bool   ArrayToFP16(
   const ushort&         dst_array[],         // 复制到
   const double&         src_array[],         // 复制自
   ENUM_FLOAT16_FORMAT   fmt                  // 格式
   );
```

参数

*dst_array[]*

[out] 接收器数组或ushort类型。

*src_array[]*

[in] float或double类型的源数组。

*fmt*

[in] 来自ENUM_FLOAT16_FORMAT枚举的复制格式。

返回值

如果成功返回true，否则返回false。

注意

格式FLOAT16和BFLOAT16在ENUM_FLOAT16_FORMAT枚举中定义，在MQL5中仅用于ONNX模型的操作。

该函数将float或double类型的输入参数转换为FLOAT16和BFLOAT16类型。然后，这些输入参数将在OnnxRun函数中使用。

FLOAT16，也称为半精度float（浮点），使用16位来表示浮点数。这种格式提供了准确性和计算效率之间的平衡。FLOAT16广泛应用于需要对大型数据集进行高性能处理的深度学习算法和神经网络。这种格式通过减少数字大小来加速计算，这在GPU上训练深度神经网络时尤其重要。

BFLOAT16（或Brain Floating Point 16）也使用16位，但在格式表示方法上与FLOAT16不同。在该格式中，分配8位用于表示指数，而剩余7位用于表示尾数。这种格式是为深度学习和人工智能而开发的，特别是在谷歌的张量处理单元(TPU)中。BFLOAT16在神经网络训练方面表现出优异的性能，可以有效加速计算。

© 2000-2025, MetaQuotes Ltd.

示例：函数来自文章 使用float16和float8格式的ONNX模型

```
//+------------------------------------------------------------------+
//| RunCastFloat16ToDouble                                           |
//+------------------------------------------------------------------+
bool RunCastFloat16ToDouble(long model_handle)
  {
   PrintFormat("test=%s",__FUNCTION__);
   double test_data[12]= {1,2,3,4,5,6,7,8,9,10,11,12};
   ushort data_uint16[12];
   if(!ArrayToFP16(data_uint16,test_data,FLOAT_FP16))
     {
      Print("error in ArrayToFP16. error code=",GetLastError());
      return(false);
     }
   Print("test array:");
   ArrayPrint(test_data);
   Print("ArrayToFP16:");
   ArrayPrint(data_uint16);
   U<ushort> input_float16_values[3*4];
   U<double> output_double_values[3*4];
   float test_data_float[];
   if(!ArrayFromFP16(test_data_float,data_uint16,FLOAT_FP16))
     {
      Print("error in ArrayFromFP16. error code=",GetLastError());
      return(false);
     }
   for(int i=0; i<12; i++)
     {
      input_float16_values[i].value=data_uint16[i];
      PrintFormat("%d input value =%f  Hex float16 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array:");
   ArrayPrint(input_float16_values);
   bool res=OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float16_values,output_double
   if(!res)
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      return(false);
     }
   Print("ONNX output array:");
   ArrayPrint(output_double_values);
//---
   double sum_error=0.0;
   for(int i=0; i<12; i++)
     {
      double delta=test_data[i]-output_double_values[i].value;
```

© 2000-2025, MetaQuotes Ltd.

```
      sum_error+=MathAbs(delta);
      PrintFormat("%d output double %f = %s  difference=%f",i,output_double_values[i],
     }
//---
   PrintFormat("test=%s   sum_error=%f",__FUNCTION__,sum_error);
//---
   return(true);
  }
```

另 见

ArrayFromFP16, ArrayCopy

© 2000-2025, MetaQuotes Ltd.

# ArrayToFP8

将float或double类型数组复制到给定格式的uchar类型的数组中。

```
bool   ArrayToFP8(
   const uchar&          dst_array[],        // 复制到
   const float&          src_array[],        // 复制自
   ENUM_FLOAT8_FORMAT    fmt                 // 格式
   );
```

double类型的重载

```
bool   ArrayToFP8(
   const uchar&          dst_array[],        // 复制到
   const double&         src_array[],        // 复制自
   ENUM_FLOAT8_FORMAT    fmt                 // 格式
   );
```

参数

*dst_array[]*

[out] 接收器数组或uchar类型。

*src_array[]*

[in] float或double类型的源数组。

*fmt*

[in] 来自ENUM_FLOAT8_FORMAT枚举的复制格式。

返回值

如果成功返回true，否则返回false。

注意

所有类型的FP8格式均在ENUM_FLOAT8_FORMAT枚举中定义，在MQL5中仅用于ONNX模型的操作。

该函数将float或double类型的输入参数转换为FP8类型之一。然后，这些输入参数将在OnnxRun函数中使用。

FP8（8位浮点）是用于表示浮点数的数据类型之一。在FP8中，每个数字由8个数据位表示，通常分为三个部分：符号、指数和尾数。这种格式提供了准确性和存储效率之间的平衡，使其对于需要内存和计算效率的应用程序具有吸引力。

通过采用紧凑型数字表示法，FP8减少了内存需求并加速了计算。此外，FP8可用于实现算术计算和信号处理等低级操作。

示例：函数来自文章使用float16和float8格式的ONNX模型

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| RunCastFloat8Float                                               |
//+------------------------------------------------------------------+
bool RunCastFloat8ToFloat(long model_handle,const ENUM_FLOAT8_FORMAT fmt)
  {
   PrintFormat("TEST: %s(%s)",__FUNCTION__,EnumToString(fmt));
//---
   float test_data[15]   = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   uchar data_float8[15] = {};
   if(!ArrayToFP8(data_float8,test_data,fmt))
     {
      Print("error in ArrayToFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   U<uchar> input_float8_values[3*5];
   U<float> output_float_values[3*5];
   float    test_data_float[];
//--- 将float8转换为float
   if(!ArrayFromFP8(test_data_float,data_float8,fmt))
     {
      Print("error in ArrayFromFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   for(uint i=0; i<data_float8.Size(); i++)
     {
      input_float8_values[i].value=data_float8[i];
      PrintFormat("%d input value =%f  Hex float8 = %s  ushort value=%d",i,test_data_f
     }
   Print("ONNX input array: ",ArrayToString(input_float8_values));
//--- 执行模型（使用ONNX将float8转换为float）
   if(!OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float8_values,output_float_values
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   Print("ONNX output array: ",ArrayToString(output_float_values));
//--- 计算误差（比较ONNX和ArrayFromFP8结果）
   double sum_error=0.0;
   for(uint i=0; i<test_data.Size(); i++)
     {
      double delta=test_data_float[i]-(double)output_float_values[i].value;
      sum_error+=MathAbs(delta);
      PrintFormat("%d output float %f = %s difference=%f",i,output_float_values[i].val
     }
//---
   PrintFormat("%s(%s): sum_error=%f\n",__FUNCTION__,EnumToString(fmt),sum_error);
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
```

另 见

ArrayFromFP8, ArrayCopy

© 2000-2025, MetaQuotes Ltd.

# ArrayFromFP16

将ushort类型数组复制到给定格式的float或double类型的数组中。

```
bool   ArrayFromFP16(
   const float&          dst_array[],        // 复制到
   const ushort&         src_array[],        // 复制自
   ENUM_FLOAT16_FORMAT   fmt                 // 格式
   );
```

double类型的重载

```
bool   ArrayFromFP16(
   const double&         dst_array[],        // 复制到
   const ushort&         src_array[],        // 复制自
   ENUM_FLOAT16_FORMAT   fmt                 // 格式
   );
```

参数

*dst_array[]*

[out] 接收器数组或float或double类型。

*src_array[]*

[in] ushort类型的源数组。

*fmt*

[in] 来自ENUM_FLOAT16_FORMAT枚举的复制格式。

返回值

如果成功返回true，否则返回false。

注意

格式FLOAT16和BFLOAT16在ENUM_FLOAT16_FORMAT枚举中定义，在MQL5中仅用于ONNX模型的操作。

如果从OnnxRun函数执行得到的输出参数为FLOAT16和BFLOAT16类型，则可以使用该函数将结果转换为float或double数组。

FLOAT16，也称为半精度float（浮点），使用16位来表示浮点数。这种格式提供了准确性和计算效率之间的平衡。FLOAT16广泛应用于需要对大型数据集进行高性能处理的深度学习算法和神经网络。这种格式通过减少数字大小来加速计算，这在GPU上训练深度神经网络时尤其重要。

BFLOAT16（或Brain Floating Point 16）也使用16位，但在格式表示方法上与FLOAT16不同。在该格式中，分配8位用于表示指数，而剩余7位用于表示尾数。这种格式是为深度学习和人工智能而开发的，特别是在谷歌的张量处理单元(TPU)中。BFLOAT16在神经网络训练方面表现出优异的性能，可以有效加速计算。

© 2000-2025, MetaQuotes Ltd.

示例：函数来自文章 使用float16和float8格式的ONNX模型

```
//+------------------------------------------------------------------+
//| RunCastFloat16ToDouble                                           |
//+------------------------------------------------------------------+
bool RunCastFloat16ToDouble(long model_handle)
  {
   PrintFormat("test=%s",__FUNCTION__);
   double test_data[12]= {1,2,3,4,5,6,7,8,9,10,11,12};
   ushort data_uint16[12];
   if(!ArrayToFP16(data_uint16,test_data,FLOAT_FP16))
     {
      Print("error in ArrayToFP16. error code=",GetLastError());
      return(false);
     }
   Print("test array:");
   ArrayPrint(test_data);
   Print("ArrayToFP16:");
   ArrayPrint(data_uint16);
   U<ushort> input_float16_values[3*4];
   U<double> output_double_values[3*4];
   float test_data_float[];
   if(!ArrayFromFP16(test_data_float,data_uint16,FLOAT_FP16))
     {
      Print("error in ArrayFromFP16. error code=",GetLastError());
      return(false);
     }
   for(int i=0; i<12; i++)
     {
      input_float16_values[i].value=data_uint16[i];
      PrintFormat("%d input value =%f  Hex float16 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array:");
   ArrayPrint(input_float16_values);
   bool res=OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float16_values,output_double
   if(!res)
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      return(false);
     }
   Print("ONNX output array:");
   ArrayPrint(output_double_values);
//---
   double sum_error=0.0;
   for(int i=0; i<12; i++)
     {
      double delta=test_data[i]-output_double_values[i].value;
```

© 2000-2025, MetaQuotes Ltd.

```
      sum_error+=MathAbs(delta);
      PrintFormat("%d output double %f = %s  difference=%f",i,output_double_values[i],
     }
//---
   PrintFormat("test=%s   sum_error=%f",__FUNCTION__,sum_error);
//---
   return(true);
  }
```

另 见

ArrayToFP16, ArrayCopy

© 2000-2025, MetaQuotes Ltd.

# ArrayFromFP8

将uchar类型数组复制到给定格式的float或double类型的数组中。

```
bool   ArrayFromFP8(
   const float&          dst_array[],        // 复制到
   const uchar&          src_array[],        // 复制自
   ENUM_FLOAT8_FORMAT    fmt                 // 格式
   );
```

double类型的重载

```
bool   ArrayFromFP8(
   const double&         dst_array[],        // 复制到
   const uchar&          src_array[],        // 复制自
   ENUM_FLOAT8_FORMAT    fmt                 // 格式
   );
```

参数

*dst_array[]*

[out] 接收器数组或float或double类型。

*src_array[]*

[in] uchar类型的源数组。

*fmt*

[in] 来自ENUM_FLOAT8_FORMAT枚举的复制格式。

返回值

如果成功返回true，否则返回false。

注意

所有类型的FP8格式均在ENUM_FLOAT8_FORMAT枚举中定义，在MQL5中仅用于ONNX模型的操作。

如果从OnnxRun函数执行得到的输出参数为来自ENUM_FLOAT8_FORMAT枚举的FP8，则可以使用该函数将结果转换为float或double数组。

FP8（8位浮点）是用于表示浮点数的数据类型之一。在FP8中，每个数字由8个数据位表示，通常分为三个部分：符号、指数和尾数。这种格式提供了准确性和存储效率之间的平衡，使其对于需要内存和计算效率的应用程序具有吸引力。

通过采用紧凑型数字表示法，FP8减少了内存需求并加速了计算。此外，FP8可用于实现算术计算和信号处理等低级操作。

示例：函数来自文章 使用float16和float8格式的ONNX模型

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| RunCastFloat8Float                                               |
//+------------------------------------------------------------------+
bool RunCastFloat8ToFloat(long model_handle,const ENUM_FLOAT8_FORMAT fmt)
  {
   PrintFormat("TEST: %s(%s)",__FUNCTION__,EnumToString(fmt));
//---
   float test_data[15]   = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   uchar data_float8[15] = {};
   if(!ArrayToFP8(data_float8,test_data,fmt))
     {
      Print("error in ArrayToFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   U<uchar> input_float8_values[3*5];
   U<float> output_float_values[3*5];
   float    test_data_float[];
//--- 将float8转换为float
   if(!ArrayFromFP8(test_data_float,data_float8,fmt))
     {
      Print("error in ArrayFromFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   for(uint i=0; i<data_float8.Size(); i++)
     {
      input_float8_values[i].value=data_float8[i];
      PrintFormat("%d input value =%f  Hex float8 = %s  ushort value=%d",i,test_data_f
     }
   Print("ONNX input array: ",ArrayToString(input_float8_values));
//--- 执行模型（使用ONNX将float8转换为float）
   if(!OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float8_values,output_float_values
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   Print("ONNX output array: ",ArrayToString(output_float_values));
//--- 计算误差（比较ONNX和ArrayFromFP8结果）
   double sum_error=0.0;
   for(uint i=0; i<test_data.Size(); i++)
     {
      double delta=test_data_float[i]-(double)output_float_values[i].value;
      sum_error+=MathAbs(delta);
      PrintFormat("%d output float %f = %s difference=%f",i,output_float_values[i].val
     }
//---
   PrintFormat("%s(%s): sum_error=%f\n",__FUNCTION__,EnumToString(fmt),sum_error);
```

© 2000-2025, MetaQuotes Ltd.

```
   return(true);
  }
```

另 见

ArrayToFP8, ArrayCopy

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量

矩阵是一个由双精度、浮点或复数组成的二维数组。

向量是一个由双精度、浮点或复数组成的一维数组。 向量没有指出它是垂直的还是水平的。它的判定应来自应用的上下文。例如，向量运算不会假设左侧向量是水平的，右侧向量是垂直的。如果需要类型指示，则可以采用一行或一列的矩阵。然而，这通常没有必要。

矩阵和向量为数据动态分配内存。事实上，矩阵和向量是具有某些属性的对象，诸如它们包含的数据类型和维度。 矩阵和向量属性可以调用 vector_a.Size()、Matrix_b.Rows()、vector_c.Norm ()、matrix_d.Cond() 等方法获取。任何维度都可以更改。

在创建和初始化矩阵时，采用的是所谓的静态方法（这些方法就像类的静态方法）。 例如：matrix::Eye(), matrix::Identity(), matrix::Ones(), vector::Ones(), matrix: :Zeros(), vector::Zeros(), matrix::Full(), vector::Full(), matrix::Tri()。

此刻，矩阵和向量运算并不意味着运用复杂的数据类型，因为这一发展方向尚未完成。

MQL5 支持将矩阵和向量传递至 DLL。这样就能够从外部变量导入相关类型的函数。

矩阵和向量以指向缓冲区的指针形式被传递到 DLL。例如，若要传递浮点类型的矩阵，从 DLL 导出的函数，其相应参数必须采用浮点类型的缓冲区指针。

MQL5

```
#import "mmlib.dll"
bool sgemm(uint flags, matrix<float> &C, const matrix<float> &A, const matrix<float> &
#import
```

C++

```
extern "C" __declspec(dllexport) bool sgemm(UINT flags, float *C, const float *A, cons
```

除了缓冲区之外，还应该传递矩阵和向量的大小，方可进行正确处理。

下面按字母顺序列出了所有矩阵和向量方法。

| 函数 | 动作 | 分类 |
|---|---|---|
| Activation | 计算激活函数值，并将其写入传递的向量/矩阵 | 机器学习 |
| ArgMax | 返回最大值的索引 | 统计 |
| ArgMin | 返回最小值的索引 | 统计 |
| ArgSort | 返回排序后的索引 | 操纵 |
| Assign | 复制矩阵、向量或数组，并自动转换 | 初始化 |
| Average | 计算矩阵/向量值的加权平均值 | 统计 |
| Cholesky | 计算乔列斯基（Cholesky）分解 | 变换 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 动作 | 分类 |
| --- | --- | --- |
| Clip | 将矩阵/向量的元素限定到给定的有效值范围 | 操纵 |
| Col | 返回列向量。 将向量写入指定的列。 | 操纵 |
| Cols | 返回矩阵中的列数 | 特征 |
| Compare | 以指定的精度比较两个矩阵/向量的元素 | 操纵 |
| CompareByDigits | 将两个矩阵/向量的元素按有效数字精度进行比较 | 操纵 |
| Cond | 计算一个矩阵的条件数 | 特征 |
| Convolve | 返回两个向量的离散线性卷积 | 乘积 |
| Copy | 返回给定矩阵/向量的副本 | 操纵 |
| Concat | 将两个子矩阵连接到一个矩阵。将两个向量连接到一个向量。 | 操纵 |
| CopyIndicatorBuffer | 将指定数量中指定指标缓冲区的数据获取到向量 | 初始化 |
| CopyRates | 获取指定品种、周期、数量的历史序列，格式为 MqlRates 结构，并存储到矩阵或向量 | 初始化 |
| CopyTicks | 从 MqlTick 结构获取跳价，存储到矩阵或向量中 | 初始化 |
| CopyTicksRange | 从 MqlTick 结构获取跳价，以指定数据范围存储到矩阵或向量中 | 初始化 |
| CorrCoef | 计算皮尔逊（Pearson）相关系数（线性相关系数） | 乘积 |
| Correlate | 计算两个向量的互相关性 | 乘积 |
| Cov | 计算协方差矩阵 | 乘积 |
| CumProd | 返回矩阵/向量元素的累积乘积，包括沿给定轴的累积乘积 | 统计 |
| CumSum | 返回矩阵/矢量元素的累积总和，包括沿给定轴的矩阵/矢量元素 | 统计 |
| Derivative | 计算激活函数导数值，并将其写入所传递的向量/矩阵 | 机器学习 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 动作 | 分类 |
| --- | --- | --- |
| Det | 计算平方可逆矩阵的行列式 | 特征 |
| Diag | 提取对角线或构造对角矩阵 | 操纵 |
| Dot | 两个向量的点积 | 乘积 |
| Eig | 计算方阵的本征值和右本征向量 | 变换 |
| EigVals | 计算一般矩阵的本征值 | 变换 |
| Eye | 返回一个矩阵，对角线上为1，其它地方为0 | 初始化 |
| Fill | 用指定值填充现有矩阵或向量 | 初始化 |
| Flat | 通过一个索引替代两个索引访问矩阵元素 | 操纵 |
| Full | 创建并返回新矩阵，并以给定值填充 | 初始化 |
| GeMM | GeMM（通用矩阵乘法）方法实现了两个矩阵的一般乘法 | 乘积 |
| HasNan | 返回矩阵/向量中NaN值的数量 | 操纵 |
| Hsplit | 将矩阵水平拆分为多个子矩阵。与axis=0的拆分相同 | 操纵 |
| Identity | 创建指定大小的单位矩阵 | 初始化 |
| Init | 矩阵或向量初始化 | 初始化 |
| Inner | 两个矩阵的内积 | 乘积 |
| Inv | 用乔丹-高斯（Jordan-Gauss）方法计算平方可逆矩阵的乘法逆 | 解法 |
| Kron | 返回两个矩阵、矩阵和向量，向量和矩阵、或两个向量的克罗内克（Kronecker）乘积 | 乘积 |
| LinearRegression | 计算包含计算的线性回归值的向量/矩阵 | 统计 |
| Loss | 计算损失函数值，并将其写入所传递的向量/矩阵 | 机器学习 |
| LstSq | 返回线性代数方程的最小二乘解（对于非二乘或退化矩阵） | 解法 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 动作 | 分类 |
| --- | --- | --- |
| LU | 实现矩阵的 LU 分解：下三角矩阵和上三角矩阵的乘积 | 变换 |
| LUP | 以部分排列实现 LUP 分解，这仅指具有行排列的 LU 分解：PA=LU | 变换 |
| MatMul | 两个矩阵的矩阵乘积 | 乘积 |
| Max | 返回矩阵/向量中的最大值 | 统计 |
| Mean | 计算元素值的算术平均值 | 统计 |
| Median | 计算矩阵/向量元素的中位数 | 统计 |
| Min | 返回矩阵/向量中的最小值 | 统计 |
| Norm | 返回矩阵或向量范数 | 特征 |
| Ones | 创建并返回一个新矩阵，其中以 1 填充 | 初始化 |
| Outer | 计算两个矩阵或两个向量的外积 | 乘积 |
| Percentile | 返回矩阵/向量元素，或沿指定轴的元素值的指定百分位数 | 统计 |
| PInv | 遵照摩尔-彭罗斯 (Moore-Penrose)方法计算矩阵的伪逆 | 解法 |
| Power | 将方阵提高整数幂 | 乘积 |
| Prod | 返回矩阵/向量元素的乘积，也可以针对给定轴执行 | 统计 |
| Ptp | 返回矩阵/向量或给定矩阵轴的数值范围 | 统计 |
| QR | 计算矩阵的 QR 分解 | 变换 |
| Quantile | 返回矩阵/向量元素，或沿指定轴的元素值的指定分位数 | 统计 |
| Random | 静态函数。创建并返回一个填充了随机值的新矩阵或向量。随机值在指定范围内均匀生成。 | 初始化 |
| Rank | 返回利用高斯 (Gaussian)方法的矩阵秩 | 特征 |
| RegressionMetric | 在指定数据数组上构建回归线，并计算回归量程作为偏差误差 | 统计 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 动作 | 分类 |
| --- | --- | --- |
| Reshape | 在不更改矩阵数据的情况下更改矩阵的形状 | 操纵 |
| Resize | 返回形状和大小均已更改的新矩阵 | 操纵 |
| Row | 返回行向量。 将向量写入指定行 | 操纵 |
| Rows | 返回矩阵中的行数 | 特征 |
| Set | 根据指定索引设置向量元素值 | 操纵 |
| Size | 返回向量的大小 | 特征 |
| SLogDet | 计算矩阵行列式的符号和对数 | 特征 |
| Solve | 求解线性矩阵方程或线性代数方程组 | 解法 |
| Sort | 按位置排序 | 操纵 |
| Spectrum | 计算矩阵的谱作为其本征值的集合，来自乘积 AT*A | 特征 |
| Split | 将矩阵拆分为多个子矩阵 | 操纵 |
| Std | 返回矩阵/向量元素，或沿指定轴的元素值的标准偏差 | 统计 |
| Sum | 返回矩阵/矢量元素的总和，也可以针对给定的轴（多轴）执行 | 统计 |
| SVD | 奇异值分解 | 变换 |
| SwapCols | 交换矩阵中的列 | 操纵 |
| SwapRows | 交换矩阵中的行 | 操纵 |
| Trace | 返回矩阵对角线的总和 | 特征 |
| Transpose | 转置（轴交换）并返回修改后的矩阵 | 操纵 |
| Tri | 构造一个矩阵，在指定的对角线上及以下为 1，在其它地方为零 | 初始化 |
| TriL | 返回矩阵的副本，其中第 k 个对角线以上的元素归零。 下三角矩阵 | 操纵 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 动作 | 分类 |
| --- | --- | --- |
| TriU | 返回矩阵的副本，其中第 k 个对角线下方的元素归零。 上三角矩阵 | 操纵 |
| Var | 计算矩阵/向量元素值的方差 | 统计 |
| Vsplit | 将矩阵垂直拆分为多个子矩阵。axis=1 的拆分相同 | 操纵 |
| Zeros | 创建并返回一个新矩阵，填充零 | 初始化 |

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量类型

矩阵和向量是 MQL5 中支持线性代数运算的特殊数据类型。存在以下数据类型：

- matrix —包含双精度元素的矩阵。
- matrixf —包含浮点元素的矩阵。
- matrixc —包含复数元素的矩阵。
- vector —包含双精度元素的向量。
- vectorf —包含浮点元素的向量。
- vectorc —包含复数元素的向量。

模板函数支持 matrix<double>、matrix<float>、vector<double>、vector<float> 等表示法，替代相应的类型。

矩阵和向量初始化方法

| 函数 | 动作 |
| --- | --- |
| Eye | 返回一个矩阵，对角线上为 1，其它地方为 0 |
| Identity | 创建指定大小的单位矩阵 |
| Ones | 创建并返回一个新矩阵，其中以 1 填充 |
| Zeros | 创建并返回一个新矩阵，填充零 |
| Full | 创建并返回新矩阵，并按给定值填充 |
| Tri | 构造一个矩阵，其中位于给定对角线为 一，而其它地方为零 |
| Init | 初始化矩阵或向量 |
| Fill | 用指定值填充现有矩阵或向量 |

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量运算的枚举

本章节说明各种矩阵和矢量方法中所用的枚举。

### ENUM_AVERAGE_MODE

平滑类型枚举。

| ID | 描述 |
|---|---|
| AVERAGE_NONE | 不平均。为每个标签分别提供结果 |
| AVERAGE_BINARY | 标签1结果（二元分类） |
| AVERAGE_MICRO | 平均误差矩阵结果（混淆矩阵） |
| AVERAGE_MACRO | 每个标签误差矩阵结果的平均结果 |
| AVERAGE_WEIGHTED | 加权平均结果 |

### ENUM_VECTOR_NORM

向量范数的枚举 vector::Norm。

| ID | 说明 |
|---|---|
| VECTOR_NORM_INF | Inf 范数 |
| VECTOR_NORM_MINUS_INF | 负 Inf 范数 |
| VECTOR_NORM_P | 范数 P |

### ENUM_MATRIX_NORM

矩阵范数的枚举 matrix::Norm，和获得的 matrix::Cond 矩阵条件数字。

| ID | 说明 |
|---|---|
| MATRIX_NORM_FROBENIUS | 弗罗贝尼乌斯（Frobenius）范数 norm |
| MATRIX_NORM_SPECTRAL | 谱范数 |
| MATRIX_NORM_NUCLEAR | 原核范数 |
| MATRIX_NORM_INF | Inf 范数 |
| MATRIX_NORM_P1 | P1 范数 |
| MATRIX_NORM_P2 | P2 范数 |

© 2000-2025, MetaQuotes Ltd.

| ID | 说明 |
| --- | --- |
| MATRIX_NORM_MINUS_INF | 负 Inf 范数 |
| MATRIX_NORM_MINUS_P1 | 负 P1 范数 |
| MATRIX_NORM_MINUS_P2 | 负 P2 范数 |

### ENUM_VECTOR_CONVOLVE

卷积枚举 vector::Convolve，和互相关 vector::Correlate。

| ID | 说明 |
| --- | --- |
| VECTOR_CONVOLVE_FULL | 卷积满 |
| VECTOR_CONVOLVE_SAME | 卷积相同 |
| VECTOR_CONVOLVE_VALID | 卷积有效 |

### ENUM_REGRESSION_METRIC

回归矩阵枚举 vector::RegressionMetric。

| ID | 说明 |
| --- | --- |
| REGRESSION_MAE | 绝对平均误差 |
| REGRESSION_MSE | 均方误差 |
| REGRESSION_RMSE | 均方根误差 |
| REGRESSION_R2 | R 平方 |
| REGRESSION_MAPE | 平均绝对百分比误差 |
| REGRESSION_MSPE | 均方百分比误差 |
| REGRESSION_RMSLE | 均方根对数误差 |
| REGRESSION_SMAPE | 对称平均绝对百分比误差 |
| REGRESSION_MAXE | 最大绝对误差 |
| REGRESSION_MEDE | 中值绝对误差 |
| REGRESSION_MPD | 平均泊松偏差 |
| REGRESSION_MGD | 平均伽玛偏差 |
| REGRESSION_EXPV | 解释方差 |

© 2000-2025, MetaQuotes Ltd.

### ENUM_CLASSIFICATION_METRIC

分类问题的指标枚举。

| ID | 描述 |
| --- | --- |
| CLASSIFICATION_ACCURACY | 所有类别的预测精度而言的模型质量 |
| CLASSIFICATION_AVERAGE_PRECISION | 平均模型精度 |
| CLASSIFICATION_BALANCED_ACCURACY | 平衡预测精度 |
| CLASSIFICATION_F1 | F1评分。模型精度和召回率之间的调和平均值 |
| CLASSIFICATION_JACCARD | Jaccard评分 |
| CLASSIFICATION_PRECISION | 预测目标类别真正值的模型准确性 |
| CLASSIFICATION_RECALL | 模型完整性 |
| CLASSIFICATION_ROC_AUC | 误差曲线下的面积 |
| CLASSIFICATION_TOP_K_ACCURACY | 正确标签出现在k个预测标签顶部的频率 |

### ENUM_LOSS_FUNCTION

损失函数计算的枚举 vector::Loss。

| ID | 说明 |
| --- | --- |
| LOSS_MSE | 均方根误差 |
| LOSS_MAE | 平均绝对误差 |
| LOSS_CCE | Categorical Crossentropy |
| LOSS_BCE | Binary Crossentropy |
| LOSS_MAPE | 平均绝对百分比误差 |
| LOSS_MSLE | 均方对数误差 |
| LOSS_KLD | 库尔巴克-莱布勒（Kullback-Leibler）背离 |
| LOSS_COSINE | 余弦相似性/邻近性 |
| LOSS_POISSON | 泊松（Poisson） |
| LOSS_HINGE | 铰链 |
| LOSS_SQ_HINGE | 方形铰链 |
| LOSS_CAT_HINGE | 分类铰链 |
| LOSS_LOG_COSH | 双曲余弦的对数 |
| LOSS_HUBER | 胡贝尔（Huber） |

© 2000-2025, MetaQuotes Ltd.

### ENUM_ACTIVATION_FUNCTION

激活函数的枚举 vector::Activation，和激活函数导数 vector::Derivative。

| ID | 说明 |
| --- | --- |
| AF_ELU | 指数线性单位 |
| AF_EXP | 指数 |
| AF_GELU | 高斯误差线性单位 |
| AF_HARD_SIGMOID | 硬 Sigmoid |
| AF_LINEAR | 线性 |
| AF_LRELU | 泄漏整流线性单元 |
| AF_RELU | 重新定义的线性单元 |
| AF_SELU | 缩放指数线性单位 |
| AF_SIGMOID | 西格玛（Sigmoid） |
| AF_SOFTMAX | Softmax |
| AF_SOFTPLUS | Softplus |
| AF_SOFTSIGN | Softsign |
| AF_SWISH | Swish |
| AF_TANH | 双曲正切函数 |
| AF_TRELU | 阈值线性修正单元 |

### ENUM_SORT_MODE

Sort 函数的排序类型枚举。

| ID | 说明 |
| --- | --- |
| SORT_ASCENDING | 升序排序 |
| SORT_DESCENDING | 降序排序 |

### ENUM_MATRIX_AXIS

用于在矩阵所有统计函数中指定轴线的枚举。

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| AXIS_NONE | 没有指定坐标轴。对所有矩阵元素执行计算，类似向量(请参见Flat方法)。 |
| AXIS_HORZ | 横轴 |
| AXIS_VERT | 竖轴 |

© 2000-2025, MetaQuotes Ltd.

# 初始化

有若干种方法可以声明和初始化矩阵和向量。

| 函数 | 动作 |
| --- | --- |
| Assign | 复制矩阵、向量或数组，并自动转换 |
| CopyIndicatorBuffer | 将指定数量中指定指标缓冲区的数据获取到向量 |
| CopyRates | 获取指定品种、周期、数量的历史序列，格式为 MqlRates 结构，并存储到矩阵或向量 |
| CopyTicks | 从 MqlTick 结构获取跳价，存储到矩阵或向量中 |
| CopyTicksRange | 从 MqlTick 结构获取跳价，以指定数据范围存储到矩阵或向量中 |
| Eye | 返回一个矩阵，对角线上为 1，其它地方为 0 |
| Identity | 创建指定大小的单位矩阵 |
| Ones | 创建并返回一个新矩阵，其中以 1 填充 |
| Zeros | 创建并返回一个新矩阵，填充零 |
| Full | 创建并返回新矩阵，并按给定值填充 |
| Tri | 构造一个矩阵，其中位于给定对角线为一，而其它地方为零 |
| Init | 初始化矩阵或向量 |
| Fill | 用指定值填充现有矩阵或向量 |
| Random | 静态函数。创建并返回一个填充了随机值的新矩阵或向量。随机值在指定范围内均匀生成。 |

声明且未指定大小（没有为数据分配内存）：

```
matrix          matrix_a;   // 双精度型矩阵
matrix<double>  matrix_a1;  // 声明双矩阵的另一种途径;可以在模板中使用
matrixf         matrix_a2;  // 浮点型矩阵
matrix<float>   matrix_a3;  // 浮点型矩阵
vector          vector_a;   // 双精度向量
vector<double>  vector_a1;
vectorf         vector_a2;  // 双精度向量
vector<float>   vector_a3;
```

声明且指定大小（为数据分配内存，但未初始化）：

```
matrix          matrix_a(128,128);          // 参数既可是常数
matrix<double>  matrix_a1(InpRows,InpCols); // 亦或变量
matrixf         matrix_a2(1,128);           // 水平向量的模拟
matrix<float>   matrix_a3(InpRows,1);       // 垂直向量的模拟
vector          vector_a(256);
```

© 2000-2025, MetaQuotes Ltd.

```
  vector<double> vector_a1(InpSize);
  vectorf        vector_a2(SomeFunc());       // 函数 SomeFunc 返回一个 ulong 型的数字,用
  vector<float>  vector_a3(InpSize+16);       // 表达式可以用作参数
```

声明且初始化（矩阵和向量大小由初始化序列确定）：

```
  matrix         matrix_a={{0.1,0.2,0.3},{0.4,0.5,0.6}};
  matrix<double> matrix_a1=matrix_a;                      // 必须是相同类型的矩阵
  matrixf        matrix_a2={{1,0,0},{0,1,0},{0,0,1}};
  matrix<float>  matrix_a3={{1,2},{3,4}};
  vector         vector_a={-5,-4,-3,-2,-1,0,1,2,3,4,5};
  vector<double> vector_a1={1,5,2.4,3.3};
  vectorf        vector_a2={0,1,2,3};
  vector<float>  vector_a3=vector_a2;                     // 必须是相同类型的向量
```

声明且初始化：

```
template<typename T>
void MatrixArange(matrix<T> &mat,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<mat.Rows(); i++)
     {
      for(ulong j=0; j<mat.Cols(); j++,value+=step)
         mat[i][j]=value;
     }
  }
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
...

  matrix  matrix_a(size_m,size_k,MatrixArange,-M_PI,0.1); // 首先创建一个大小 size_m x s
  matrixf matrix_a1(10,20,MatrixArange);                  // 创建矩阵后，调用带有默认参数的
  vector  vector_a(size,VectorArange,-10.0);              // 创建向量后，调用带有一个参数的
  vectorf vector_a1(128,VectorArange);
```

请注意，矩阵或矢量大小可以更改，因为数据的内存始终是动态的。

## 静态方法

依据指定大小创建矩阵和向量的静态方法，并以某种方式初始化：

```
  matrix         matrix_a =matrix::Eye(4,5,1);
```

© 2000-2025, MetaQuotes Ltd.

```
matrix<double> matrix_a1=matrix::Full(3,4,M_PI);
matrixf        matrix_a2=matrixf::Identity(5,5);
matrixf<float> matrix_a3=matrixf::Ones(5,5);
matrix         matrix_a4=matrix::Tri(4,5,-1);
vector         vector_a =vector::Ones(256);
vectorf        vector_a1=vector<float>::Zeros(16);
vector<float>  vector_a2=vectorf::Full(128,float_value);
```

已创建矩阵和向量的初始化方法：

```
matrix  matrix_a;
matrix_a.Init(size_m,size_k,MatrixArange,-M_PI,0.1);
matrixf matrix_a1(3,4);
matrix_a1.Init(10,20,MatrixArange);
vector  vector_a;
vector_a.Init(128,VectorArange);
vectorf vector_a1(10);
vector_a1.Init(vector_size,VectorArange,start_value,step);

matrix_a.Fill(double_value);
vector_a1.Fill(FLT_MIN);
matrix_a1.Identity();
```

© 2000-2025, MetaQuotes Ltd.

# Assign

复制矩阵、向量或数组，且带有自动转换。

```
bool matrix::Assign(
  const matrix<T>  &mat     // 复制的矩阵
   );
bool matrix::Assign(
  const void       &array[] // 复制的数组
   );
bool vector::Assign(
  const vector<T>  &vec     // 复制的向量
   );
bool vector::Assign(
  const void       &array[] // 复制的数组
   );
```

参数

*m, v 或 array*

[输入] 需复制的源矩阵、向量或数组。

返回值

如果成功则返回 true，否则返回 —false。

注意

不同于 Copy，Assign 方法还允许复制数组。 在这种情况下，将进行自动转换，同时生成的矩阵或向量会根据复制数组的大小进行调整。

示例：

```
//--- 复制矩阵
  matrix a= {{2, 2}, {3, 3}, {4, 4}};
  matrix b=a+2;
  matrix c;
  Print("matrix a \n", a);
  Print("matrix b \n", b);
  c.Assign(b);
  Print("matrix c \n", a);

//--- 复制数组到矩阵
  matrix double_matrix=matrix::Full(2,10,3.14);
  Print("double_matrix before Assign() \n", double_matrix);
  int int_arr[5][5]= {{1, 2}, {3, 4}, {5, 6}};
  Print("int_arr: ");
  ArrayPrint(int_arr);
  double_matrix.Assign(int_arr);
```

© 2000-2025, MetaQuotes Ltd.

```
Print("double_matrix after Assign(int_arr) \n", double_matrix);
/*
 matrix a
 [[2,2]
  [3,3]
  [4,4]]
 matrix b
 [[4,4]
  [5,5]
  [6,6]]
 matrix c
 [[2,2]
  [3,3]
  [4,4]]

 double_matrix before Assign()
 [[3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14]
  [3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14]]

 int_arr:
     [,0][,1][,2][,3][,4]
 [0,]   1   2   0   0   0
 [1,]   3   4   0   0   0
 [2,]   5   6   0   0   0
 [3,]   0   0   0   0   0
 [4,]   0   0   0   0   0

 double_matrix after Assign(int_arr)
 [[1,2,0,0,0]
  [3,4,0,0,0]
  [5,6,0,0,0]
  [0,0,0,0,0]
  [0,0,0,0,0]]

*/
```

参见

Copy

© 2000-2025, MetaQuotes Ltd.

# CopyIndicatorBuffer

将指定数量中指定指标缓冲区的数据获取到向量。

数据将被复制到向量中，定位到为该向量分配的物理内存开头的最早元素。共有三个函数选项。

根据初始位置和所需元素的数量进行访问

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // 指标句柄
   ulong     buffer_index,         // 指标缓冲区编号
   ulong     start_pos,            // 要复制的开始位置
   ulong     count                 // 要复制的元素数量
   );
```

根据开始日期和所需元素的数量进行访问

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // 指标句柄
   ulong     buffer_index,         // 指标缓冲区编号
   datetime  start_time,           // 复制的开始日期
   ulong     count                 // 复制的元素数量
   );
```

根据所需时间间隔的初始日期和最终日期进行访问

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // 指标句柄
   ulong     buffer_index,         // 指标缓冲区编号
   datetime  start_time,           // 复制的开始日期
   datetime  stop_time             // 复制的结束日期
   );
```

参数

*indicator_handle*

[in] 由相关指标函数获得的指标句柄。

*buffer_index*

[in] 指标缓存区的编号。

*start_pos*

[in] 第一个复制元素的索引编号。

*count*

[in] 复制的元素数。

*start_time*

[in] 第一个元素对应的柱形图时间。

*stop_time*

[in] 最后一个元素对应的柱形图时间。

© 2000-2025, MetaQuotes Ltd.

返回值

如果成功，函数返回'true'，如果出现错误则返回'false'。

注意

复制数据的元素（索引为buffer_index的指标缓冲区）从现在到过去进行计数，因此起始位置等于0表示当前柱形图（当前柱形图的指标值）。

当复制未知数量的数据时，您应该声明一个不指定大小的向量（不为数据分配内存），因为CopyBuffer()函数试图将接收向量的大小分配给复制数据的大小。

当需要部分复制指标值时，应该使用一个中间向量，将所需的数量复制到其中。从这个中间向量，您可以逐个成员地将所需数量的值复制到接收向量的所需位置

如果要复制预定数量的数据，建议预先声明一个向量并指定其大小，以避免不必要的内存重新分配。

从指标请求数据时，如果请求的时间序列尚未构建或需要从服务器下载，则该函数立即返回false，同时启动加载/构建。

当从EA或脚本请求数据时，如果程序端在本地没有合适的数据，则启动从服务器下载，或者如果可以从本地历史构建数据，但所需的时间序列尚未准备好，则开始构建必要的时间序列。该函数返回在超时到期时将准备好的数量。

另见

CopyBuffer

© 2000-2025, MetaQuotes Ltd.

# CopyRates

获取 MqlRates 结构的历史序列，按指定品种-周期，按指定数量，并存储到矩阵或向量当中。元素顺序倒计时，从现在到过去，这意味着起始位置等于 0 表示当前柱线。

复制数据，以便将最久远的元素放置在矩阵/向量的开头。有三个函数选项。

依据初始位置和所需元素的数量进行访问

```
bool  matrix::CopyRates(
   string           symbol,       // 品名
   ENUM_TIMEFRAMES  period,       // 周期
   ulong            rates_mask,   // 指定所请求序列的标志组合
   ulong            start,        // 复制初始柱线的起始索引
   ulong            count         // 要复制的数量
   );
```

按初始日期和所需元素的数量进行访问

```
bool  matrix::CopyRates(
   string           symbol,       // 品名
   ENUM_TIMEFRAMES  period,       // 周期
   ulong            rates_mask,   // 指定所请求序列的标志组合
   datetime         from,         // 起始日期
   ulong            count         // 要复制的数量
   );
```

按所需时间间隔的初始日期和最终日期进行访问

```
bool  matrix::CopyRates(
   string           symbol,       // 品名
   ENUM_TIMEFRAMES  period,       // 周期
   ulong            rates_mask,   // 指定所请求序列的标志组合
   datetime         from,         // 起始日期
   datetime         to            // 结束日期
   );
```

向量方法

依据初始位置和所需元素的数量进行访问

```
bool  vector::CopyRates(
   string           symbol,       // 品名
   ENUM_TIMEFRAMES  period,       // 周期
   ulong            rates_mask,   // 指定所请求序列的标志组合
   ulong            start,        // 复制初始柱线的起始索引
   ulong            count         // 要复制的数量
   );
```

按初始日期和所需元素的数量进行访问

```
bool  vector::CopyRates(
```

© 2000-2025, MetaQuotes Ltd.

```
   string            symbol,        // 品名
   ENUM_TIMEFRAMES   period,        // 周期
   ulong             rates_mask,    // 指定所请求序列的标志组合
   datetime          from,          // 起始日期
   ulong             count          // 要复制的数量
   );
```

按所需时间间隔的初始日期和最终日期进行访问

```
bool  vector::CopyRates(
   string            symbol,        // 品名
   ENUM_TIMEFRAMES   period,        // 周期
   ulong             rates_mask,    // 指定所请求序列的标志组合
   datetime          from,          // 起始日期
   datetime          to             // 结束日期
   );
```

参数

*symbol*

[输入] 品种。

*period*

[输入] 周期。

*rates_mask*

[输入] 指定所请求序列类型的标志的 ENUM_COPY_RATES 枚举组合。当复制到向量时，只能指定来自 ENUM_COPY_RATES 枚举中的一个值，否则会发生错误。

*start*

[输入] 第一个复制的元素索引。

*count*

[输入] 复制的元素数量。

*from*

[输入] 对应于第一个元素的柱线时间。

*to*

[输入] 对应于最后一个元素的柱线时间。

返回值

如果成功则返回 true，否则返回 false（如果出现错误）。

注意

如果所请求数据的区间完全超出服务器上的可用数据，则该函数返回 false。如果数据超出 TERMINAL_MAXBARS 之外（图表上的最大柱数），则该函数也会返回 false。

当请求数据来自 EA 或脚本时，如果终端在本地没有相应的数据，则启动从服务器下载，或者如果可以从本地历史构建数据但尚未准备好，则开始构建必要的时间序列。该函数按时间

© 2000-2025, MetaQuotes Ltd.

返回准备好的直至期满的数据量，然而历史记录下载将继续，并且在下一个类似请求期间，该函数会返回更多数据。

按开始日期和所需项目数量请求数据时，仅返回日期小于（之前）或等于指定日期的数据。设置间隔时要考虑多一秒。 换句话说，返回值的任何柱线的开盘日期（交易量、点差、开盘价、最高价、最低价、收盘价或时间）始终等于或小于指定的日期。

当请求给定日期范围内的数据时，仅返回请求间隔内的数据。设置间隔时要考虑多一秒。换句话说，返回的任何柱线的开盘时间（交易量、点差、指标缓冲区值、开盘价、最高价、最低价、收盘价或时间）始终位于请求的区间内。

例如，如果当日是一周中的星期六，则当尝试设置 *start_time=Last_Tuesday* 和 *stop_time=Last_Friday* 复制周线时间帧数据时，函数会返回 0，因为周线时间帧的开盘时间始终落在星期日，故没有一个周线柱线落在指定范围内。

如果您需要获取对应于当前未完成柱线的数值，那么您可以采用第一种调用形式指示 *start_pos*=0 和 *count*=1。

### ENUM_COPY_RATES

ENUM_COPY_RATES 枚举包含指定要传递给矩阵或数组的数据类型的标志。 标志组合允许在一个请求中从历史记录中获取多个序列。矩阵中行的顺序将与 ENUM_COPY_RATES 枚举中值的顺序相对应。换言之，含有最高价数据的行在矩阵中将始终高于含有最低价数据的行。

| ID | 值 | 说明 |
|---|---|---|
| COPY_RATES_OPEN | 1 | 开盘价序列 |
| COPY_RATES_HIGH | 2 | 最高价序列 |
| COPY_RATES_LOW | 4 | 最低价序列 |
| COPY_RATES_CLOSE | 8 | 收盘价序列 |
| COPY_RATES_TIME | 16 | 时间序列（柱线开盘时间）<br><br>获取向量和矩阵（vectord 和 matrixf）的浮点型的时间会导致 ~100 秒的缺失，因为浮点精度受到严重限制，并且大于1<<24的整数无法在浮点数中准确表示。 |
| COPY_RATES_VOLUME_TICK | 32 | 跳价的成交量 |
| COPY_RATES_VOLUME_REAL | 64 | 交易成交量 |
| COPY_RATES_SPREAD | 128 | 点差 |
| 组合 | | |
| COPY_RATES_OHLC | 15 | 开盘价、最高价、最低价、和收盘价序列 |
| COPY_RATES_OHLCT | 31 | 开盘价、最高价、最低价、收盘价、和时间序列 |
| 数据排列 | | |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 说明 |
| --- | --- | --- |
| COPY_RATES_VERTICAL | 32768 | 序列沿垂直轴复制到矩阵中。接收到的序列值将在矩阵中垂直排列，即最早的数据将位于第一行，而最新的数据将位于矩阵的最后一行。<br><br>使用默认复制时，序列会沿水平轴添加到矩阵中。<br><br>该标识仅在复制到矩阵时适用。 |

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- 获取报价至矩阵
  matrix matrix_rates;
  if(matrix_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_OHLCT, 1, 10))
    Print("matrix rates: \n", matrix_rates);
  else
    Print("matrix_rates.CopyRates failed. Error ", GetLastError());
//--- 检查
  MqlRates mql_rates[];
  if(CopyRates(Symbol(), PERIOD_CURRENT, 1, 10, mql_rates)>0)
   {
    Print("mql_rates array:");
    ArrayPrint(mql_rates);
   }
  else
    Print("CopyRates(Symbol(), PERIOD_CURRENT,1, 10, mql_rates). Error ", GetLastErro
//--- 获取报价至向量 = 无效调用
  vector vector_rates;
  if(vector_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_OHLC, 1, 15))
    Print("vector_rates COPY_RATES_OHLC: \n", vector_rates);
  else
    Print("vector_rates.CopyRates COPY_RATES_OHLC failed. Error ", GetLastError());
//--- 获取收盘价至向量
  if(vector_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_CLOSE, 1, 15))
    Print("vector_rates COPY_RATES_CLOSE: \n", vector_rates);
  else
    Print("vector_rates.CopyRates failed. Error ", GetLastError());
 };
/*
   matrix rates:
   [[0.99686,0.99638,0.99588,0.99441,0.99464,0.99594,0.99698,0.99758,0.99581,0.995280
    [0.99708,0.99643,0.99591,0.9955000000000001,0.99652,0.99795,0.99865,0.99764,0.996
```

© 2000-2025, MetaQuotes Ltd.

```
    [0.9961100000000001,0.99491,0.99426,0.99441,0.99448,0.99494,0.9964499999999999,0.9
    [0.99641,0.99588,0.99441,0.99464,0.99594,0.99697,0.99758,0.99581,0.9952800000000(
    [1662436800,1662440400,1662444000,1662447600,1662451200,1662454800,1662458400,166
  mql_rates array:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
  [0] 2022.09.06 04:00:00 0.99686 0.99708 0.99611 0.99641          4463        0
  [1] 2022.09.06 05:00:00 0.99638 0.99643 0.99491 0.99588          4519        0
  [2] 2022.09.06 06:00:00 0.99588 0.99591 0.99426 0.99441          3060        0
  [3] 2022.09.06 07:00:00 0.99441 0.99550 0.99441 0.99464          3867        0
  [4] 2022.09.06 08:00:00 0.99464 0.99652 0.99448 0.99594          5280        0
  [5] 2022.09.06 09:00:00 0.99594 0.99795 0.99494 0.99697          7227        0
  [6] 2022.09.06 10:00:00 0.99698 0.99865 0.99645 0.99758         10130        0
  [7] 2022.09.06 11:00:00 0.99758 0.99764 0.99472 0.99581          7012        0
  [8] 2022.09.06 12:00:00 0.99581 0.99604 0.99360 0.99528          6166        0
  [9] 2022.09.06 13:00:00 0.99528 0.99570 0.99220 0.99259          6950        0
  vector_rates.CopyRates COPY_RATES_OHLC failed. Error 4003
  vector_rates COPY_RATES_CLOSE:
  [0.9931,0.99293,0.99417,0.99504,0.9968399999999999,0.99641,0.99588,0.99441,0.99464,
*/
```

参见

访问时间序列和指标, CopyRates

© 2000-2025, MetaQuotes Ltd.

# CopyTicks

从 MqlTick 结构获取跳价数据并存储到矩阵或向量当中。元素顺序从过去到现在计数，这意味着索引为 0 的刻度是最久远的。若要分析跳价，检查 *flags* 字段，其显示跳价中的确切变化。

```
bool  matrix::CopyTicks(
   string           symbol,              // 品名
   ulong            flags,               // 指示欲接收跳价类型的标志
   ulong            from_msc,            // 欲请求跳价的起始时间
   ulong            count                // 欲接收跳价的数量
   );
```

向量方法

```
bool  vector::CopyTicks(
   string           symbol,              // 品名
   ulong            flags,               // 指示欲接收跳价类型的标志
   ulong            from_msc,            // 欲请求跳价的起始时间
   ulong            count                // 欲接收跳价的数量
   );
```

参数

*symbol*

[输入] 品种。

*flags*

[输入] ENUM_COPY_TICKS 枚举中的标志组合，指示所请求数据的内容。当复制到向量时，只能从 ENUM_COPY_TICKS 枚举中指定一个值，否则会发生错误。

*from_msc*

[输入] 请求跳价的开始时间。自 1970 年 01 月 01 日起，时间以毫秒为单位指定。如果 from_msc=0，则返回等于 'count' 的最后跳价数字。

*count*

[输入] 所请求跳价的数量。如果参数 'from_msc' 和 'count' 未指定，则将写入所有可用的跳价，但不超过 2000。

返回值

成功时返回 true，如果发生错误则 false。

注意

首次调用 CopyTicks() 会启动硬盘驱动器上存储的相关品种跳价数据库的同步。如果本地数据库没有提供全部请求的跳价，则缺失的跳价将自动从交易服务器下载。 跳价的同步自 CopyTicks() 中指定的 *from_msc* 到当前时刻。之后，该品种所有到达的跳价都将添加到跳价数据库之中，从而使其保持同步状态。

如果 *from_msc* 和 *count* 参数未指定，则把全部可用跳价，但不超过 2000 个，写入矩阵/向量之中。

© 2000-2025, MetaQuotes Ltd.

**在指标中，CopyTicks() 方法立即返回结果**：当从指标调用时，CopyTick() 会立即返回品种的所有可用跳价，如果可用数据不足，则启动跳价数据库的同步。同一品种上的所有指标都在一个公用线程中运行，因此指标不能等待同步完成。同步后，CopyTicks() 将在下一次调用期间返回所有请求的跳价。在指标中，OnCalculate() 函数在每次跳价到达后调用。

**在智能系统和脚本中，CopyTicks() 可以等待结果 45 秒**：与指标不同，每个智能系统或脚本都在单独的线程中运行，因此为完成同步最多可以等待 45 秒。如果在此期间无法同步所需数量的跳价，CopyTicks() 将在超时后返回可用的跳价，并继续同步。智能系统里的 OnTick() 并非每次跳价都会处理，它只通知智能系统行情有所变化。这可能是一批变化：终端可以同时接收多次跳价，而 OnTick() 只会调用一次，通知智能系统最新的行情状态。

**数据返回率**：终端在高速访问缓存中为每个金融产品存储了 4096 个最后的跳价（市场深度为运营品种存储 65536 个跳价）。有关此数据的请求执行速度最快。如果当前交易时段请求的跳价超出缓存，CopyTicks() 将调用存储在终端内存中的跳价。这些请求需要更多时间才能完成。最慢的请并非那些请求其它日期的跳价，因为在这种情况下，数据是从驱动器读取的。

### ENUM_COPY_TICKS

ENUM_COPY_TICKS 枚举包含指定要传递给矩阵或数组的数据类型的标志。标志组合允许在一个请求中从历史记录中获取多个序列。矩阵中行的顺序将与 ENUM_COPY_TICKS 枚举中数值的顺序相对应。换言之，含有最高价数据的行在矩阵中将始终高于含有最低价数据的行。

| ID | 值 | 说明 |
|---|---|---|
| COPY_TICKS_INFO | 1 | 包含卖价（Bid）和/或买价（Ask）价格变化的报价 |
| COPY_TICKS_TRADE | 2 | 包含最后价（Last）和/或交易量（Volume）价格变化的报价 |
| COPY_TICKS_ALL | 3 | 所有报价 |
| COPY_TICKS_TIME_MS | 1<<8 | 跳价时间（以毫秒为单位） |
| COPY_TICKS_BID | 1<<9 | 出价（Bid） |
| COPY_TICKS_ASK | 1<<10 | 要价（Ask） |
| COPY_TICKS_LAST | 1<<11 | 最后价（最后成交价） |
| COPY_TICKS_VOLUME | 1<<12 | 最后价交易量 |
| COPY_TICKS_FLAGS | 1<<13 | 跳价标志 |
| 数据排列 | | |
| COPY_TICKS_VERTICAL | 32768 | 报价沿垂直轴复制到矩阵中。接收到的报价将在矩阵中垂直排列，即最早的报价将位于第一行，而最新的报价将位于矩阵的最后一行。<br><br>使用默认复制时，报价会沿水平轴添加到矩阵中。<br><br>该标识仅在复制到矩阵时适用。 |

© 2000-2025, MetaQuotes Ltd.

分析跳价标志以便找出哪些数据已变化：

- TICK_FLAG_BID —跳价改变了出价
- TICK_FLAG_ASK —跳价改变了要价
- TICK_FLAG_LAST —跳价改变了最后成交价
- TICK_FLAG_VOLUME —跳价改变了交易量
- TICK_FLAG_BUY —跳价是买入成交的结果
- TICK_FLAG_SELL —跳价是卖出成交的结果

参见

访问时间序列和指标, CopyTicks

© 2000-2025, MetaQuotes Ltd.

# CopyTicksRange

按指定日期范围，获取 MqlTick 结构的跳价，并存储到矩阵或向量之中。元素顺序从过去到现在计数，这意味着索引为 0 的跳价是最久远的。若要分析跳价，检查 *flags* 字段，其显示跳价中的确切变化。

```
bool  matrix::CopyTicksRange(
   string            symbol,                 // 品名
   ulong             flags,                  // 指示欲接收跳价类型的标志
   ulong             from_msc,               // 欲请求跳价的起始时间
   ulong             to_msc                  // 欲请求跳价的截止时间
   );
```

向量方法

```
bool  vector::CopyTicksRange(
   string            symbol,                 // 品名
   ulong             flags,                  // 指示欲接收跳价类型的标志
   ulong             from_msc,               // 欲请求跳价的起始时间
   ulong             to_msc                  // 欲请求跳价的截止时间
   );
```

参数

*symbol*

[输入] 品种。

*flags*

[输入] ENUM_COPY_TICKS 枚举中的标志组合，指示所请求数据的内容。当复制到向量时，只能从 ENUM_COPY_TICKS 枚举中指定一个值，否则会发生错误。

*from_msc*

[输入] 请求跳价的开始时间。自 1970 年 01 月 01 日起，时间以毫秒为单位指定。如果未指定 "from_msc" 参数，则跳价从头开始返回历史记录。即返回时间 >= from_msc 的跳价。

*to_msc*

[输入] 请求截止的跳价时间。自 1970 年 01 月 01 日起，时间以毫秒为单位指定。返回时间 <= to_msc 的跳价。如果未指定 to_msc 参数，则返回截至历史记录结束之前的所有跳价。

返回值

成功时返回 true，如果发生错误则返回 false。GetLastError() 能返回以下错误：

- ERR_HISTORY_TIMEOUT —跳价同步的超时已过期，函数返回它已拥有的一切。
- ERR_HISTORY_SMALL_BUFFER —静态缓存区太小。仅返回数组可以存储的数量。
- ERR_NOT_ENOUGH_MEMORY — 内存不足，无法将指定范围的历史数据接收到动态跳价数组之中。为跳价数组分配足够内存失败。

分析跳价标志以便找出哪些数据已变化：

- TICK_FLAG_BID —跳价改变了出价

© 2000-2025, MetaQuotes Ltd.

- TICK_FLAG_ASK — 跳价改变了要价
- TICK_FLAG_LAST — 跳价改变了最后成交价
- TICK_FLAG_VOLUME — 跳价改变了交易量
- TICK_FLAG_BUY — 跳价是买入成交的结果
- TICK_FLAG_SELL — 跳价是卖出成交的结果

注意

CopyTicksRange() 方法请求的跳价精确到指定范围。 例如，历史记录中指定日期的跳价。CopyTicks() 只允许指定开始日期，例如，接收从月初到现在的所有跳价。

参见

访问时间序列和指标, CopyTicksRange

© 2000-2025, MetaQuotes Ltd.

# Eye

统计函数。构造具有指定大小的矩阵，主对角线上为 1，其它之处则为 0。返回一个矩阵，对角线上为 1，其它之处为零。

```
static matrix matrix::Eye(
  const ulong  rows,         // 行数
  const ulong  cols,         // 列数
  const int    ndiag=0       // 对角线索引
   );
```

参数

*rows*

[输入] 输出中的行数。

*cols*

[输入] 输出中的列数。

*ndiag=0*

[输入] 对角线索引：0（默认值）表示主对角线，正值表示上对角线，负值表示下对角线。

返回值

所有元素都等于零的矩阵，但第 k 对角线除外，其值等于 1。

MQL5 示例：

```
  matrix eye=matrix::Eye(3, 3);
  Print("eye = \n", eye);

  eye=matrix::Eye(4, 4,1);
  Print("eye = \n", eye);
  /*
   eye =
   [[1,0,0]
    [0,1,0]
    [0,0,1]]
   eye =
   [[0,1,0,0]
    [0,0,1,0]
    [0,0,0,1]
    [0,0,0,0]]
  */
```

Python 示例：

```
 np.eye(3, dtype=int)
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

© 2000-2025, MetaQuotes Ltd.

```
 np.eye(4, k=1)
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

© 2000-2025, MetaQuotes Ltd.

# Identity

它是一个静态函数，可创建指定大小（不一定是正方形）的单位矩阵。 一个单位矩阵在主对角线上包含 0，在其它所在包含零。 主对角线由行索引和列索引相等的矩阵元素组成，譬如 [0,0]、[1,1]、[2,2]，等等。创建一个新的单位矩阵。

还有一个 Identity 方法能将已经存在的矩阵转换为单位矩阵。

```
static matrix matrix::Identity(
  const ulong  rows,         // 行数
  const ulong  cols,         // 列数
   );

void matrix::Identity();
```

参数

*rows*

[输入] n x n 矩阵中的行数（和列数）。

返回值

返回单位矩阵。单位矩阵是一个方阵，主对角线上为 1。

MQL5 示例：

```
  matrix identity=matrix::Identity(3,3);
  Print("identity = \n", identity);
/*
   identity =
   [[1,0,0]
    [0,1,0]
    [0,0,1]]
*/
  matrix identity2(3,5);
  identity2.Identity();
  Print("identity2 = \n", identity2);
/*
   identity2 =
   [[1,0,0,0,0]
    [0,1,0,0,0]
    [0,0,1,0,0]]
*/
```

Python 示例：

```
np.identity(3)
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
```

© 2000-2025, MetaQuotes Ltd.

```
        [0.,  0.,  1.]])
```

© 2000-2025, MetaQuotes Ltd.

## Ones

这是一个静态函数，创建并返回一个新矩阵，且以 1 填充。

```
static matrix matrix::Ones(
  const ulong  rows,      // 行数
  const ulong  cols       // 列数
   );

static vector vector::Ones(
  const ulong  size,      // 向量大小
   );
```

参数

*rows*

[输入] 行数。

*cols*

[输入] 列数。

返回值

给定行数和列数的新矩阵，且填充了 1。

MQL5 示例：

```
  matrix ones=matrix::Ones(4, 4);
  Print("ones = \n", ones);
/*
ones =
   [[1,1,1,1]
    [1,1,1,1]
    [1,1,1,1]
    [1,1,1,1]]
*/
```

Python 示例：

```
np.ones((4, 1))
array([[1.],
       [1.]])
```

© 2000-2025, MetaQuotes Ltd.

# Zeros

这是一个静态函数，创建并返回一个填充零值的新矩阵。

```
static matrix matrix::Zeros(
  const ulong  rows,      // 行数
  const ulong  cols       // 列数
   );

static vector vector::Zeros(
  const ulong  size,      // 向量大小
   );
```

参数

*rows*

[输入] 行数。

*cols*

[输入] 列数。

返回值

给定行数和列数的新矩阵，并以零填充。

MQL5 示例：

```
  matrix zeros=matrix::Zeros(3, 4);
  Print("zeros = \n", zeros);
/*
zeros =
   [[0,0,0,0]
    [0,0,0,0]
    [0,0,0,0]]
*/
```

Python 示例：

```
np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

© 2000-2025, MetaQuotes Ltd.

# Full

静态函数创建并返回一个新矩阵，并按给定值填充。

```
static matrix matrix::Full(
  const ulong   rows,       // number or rows
  const ulong   cols,       // number of columns
  const double  value       // 填充值
   );

static vector vector::Full(
  const ulong   size,       // vector size
  const double  value       // 填充值
   );
```

参数

*rows*

[输入] 行数。

*cols*

[输入] 列数。

*value*

[输入] 填充所有矩阵元素的数值。

返回值

返回给定行和列的新矩阵，并用指定值填充。

MQL5 示例：

```
  matrix full=matrix::Full(3,4,10);
  Print("full = \n", full);
/*
full =
   [[10,10,10,10]
    [10,10,10,10]
    [10,10,10,10]]
*/
```

举例

```
np.full((2, 2), 10)
array([[10, 10],
       [10, 10]])
```

© 2000-2025, MetaQuotes Ltd.

# Tri

这是一个构造矩阵的静态函数，其中给定对角线为 1，，且其它地方为 0。

```
static matrix matrix::Tri(
  const ulong rows,         // 行数
  const ulong cols,         // 列数
  const int   ndiag=0       // 对角线数
   );
```

参数

*rows*

[输入] 数组中的行数。

*cols*

[输入] 数组中的列数。

*ndiag=0*

[输入] 子对角线及其下方数组被填充。k = 0 是主对角线，而 k < 0 在其下方，k > 0 在上方。默认填充值为 0。

返回值

数组，其下三角以 1 填充，在其它所在则填充 0。

MQL5 示例：

```
   matrix matrix_a=matrix::Tri(3,4,1);
   Print("Tri(3,4,1)\n",matrix_a);
   matrix_a=matrix::Tri(4,3,-1);
   Print("Tri(4,3,-1)\n",matrix_a);

/*
   Tri(3,4,1)
   [[1,1,0,0]
    [1,1,1,0]
    [1,1,1,1]]
   Tri(4,3,-1)
   [[0,0,0]
    [1,0,0]
    [1,1,0]
    [1,1,1]]
*/
```

举例

```
np.tri(3, 5, 2, dtype=int)
array([[1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0],
       [1, 1, 1, 1, 1]])
```

© 2000-2025, MetaQuotes Ltd.

## Init

初始化矩阵或向量。

```
void matrix::Init(
  const ulong  rows,            // 行数
  const ulong  cols,            // 列数
  func_name    init_func=NULL,  // 置于类的某个作用域或静态方法中的 init 函数
   ...         parameters
   );

void vector::Init(
  const ulong  size,            // 向量大小
  func_name    init_func=NULL,  // 置于类的某个作用域或静态方法中的 init 函数
   ...         parameters
   );
```

参数

*rows*

[输入] 行数。

*cols*

[输入] 列数。

*func_name*

[输入] 初始化函数。

*...*

[输入] 初始化函数的参数。

返回值

无返回值。

举例

```
template<typename T>
void MatrixArange(matrix<T> &mat,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<mat.Rows(); i++)
     {
      for(ulong j=0; j<mat.Cols(); j++,value+=step)
         mat[i][j]=value;
     }
  }
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
  int size_m=3, size_k=4;
  matrix  m(size_m,size_k,MatrixArange,-2.,0.1); // 首先创建一个大小为 size_ m x size_k
  Print("matrix m \n",m);                        // 然后调用初始化清单列出的带参数的 Matri
  matrixf m_float(5,5,MatrixArange,-2.f,0.1f);   // 创建浮点型矩阵后，调用 MatrixArange 函
  Print("matrix m_float \n",m_float);
  vector  v(size_k,VectorArange,-10.0);          // 创建一个向量后，会调用带有一个参数的 Ve
  Print("vector v \n",v);
  /*
   matrix m
   [[-2,-1.9,-1.8,-1.7]
    [-1.6,-1.5,-1.399999999999999,-1.299999999999999]
    [-1.199999999999999,-1.099999999999999,-0.9999999999999992,-0.8999999999999992]]
   matrix m_float
   [[-2,-1.9,-1.8,-1.6999999,-1.5999999]
    [-1.4999999,-1.3999999,-1.2999998,-1.1999998,-1.0999998]
    [-0.99999976,-0.89999974,-0.79999971,-0.69999969,-0.59999967]
    [-0.49999967,-0.39999968,-0.29999968,-0.19999969,-0.099999689]
    [3.1292439e-07,0.10000031,0.20000032,0.30000031,0.4000003]]
   vector v
   [-10,-9,-8,-7]
  */
  }
```

© 2000-2025, MetaQuotes Ltd.

# Fill

用指定的数值填充存在的矩阵或向量。

```
void matrix::Fill(
  const double  value      // 填充值
   );

void vector::Fill(
  const double  value      // 填充值
   );
```

参数

*value*

[输入] 填充所有矩阵元素的数值。

返回值

无返回值。矩阵将用指定的值填充。

举例

```
   matrix matrix_a(2,2);
   matrix_a.Fill(10);
   Print("matrix_a\n",matrix_a);

/*
  matrix_a
  [[10,10]
   [10,10]]
*/
```

© 2000-2025, MetaQuotes Ltd.

## Random

静态函数。创建并返回一个填充了随机值的新矩阵或向量。随机值在指定范围内均匀生成。

```
static vector vector::Random(
  const ulong   size,        // 向量长度
  const double  min=0.0,     // 最小值
  const double  max=1.0      // 最大值
   );

static matrix matrix::Random(
  const ulong   rows,        //行数
  const ulong   cols         // 列数
  const float   min=0.0,     // 最小值
  const float   max=1.0      // 最大值
   );
```

该函数用随机值填充现有的矩阵或向量。随机值在指定范围内均匀生成。

```
void  vector::Random(
  const double  min=0.0,     // 最小值
  const double  max=1.0      // 最大值
   );

void  matrix::Random(
  const float  min=0.0,      // 最小值
  const float  max=1.0       // 最大值
   );
```

参数

*rows*

[in] 行数。

*cols*

[in] 列数。

*size*

[in] 向量长度。

*min=0.0*

[in] 生成的随机数样本中的最小值。

*max=1.0*

[in] 生成的随机数样本中的最大值。

返回值

返回一个由给定行和列组成的新矩阵，其中填充了随机值。

© 2000-2025, MetaQuotes Ltd.

示例

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector v1= vector::Random(6,0,4);
   Print("vector v1 \n", v1);
   /*
   vector v1
   [3.340834286841758,0.9073984578501895,3.15598511117417,0.7106475841045956,2.294010(
   */

   vector v2(4);
   v2.Random(-3,3);
   Print("vector v2 \n", v2);
   /*
   vector v2
   [-2.642032080246609,1.008521607147651,1.340623848219547,-2.789016363161853]
   */

   matrix m1 = matrix::Random(5, 5, -1, 3);
   Print("matrix m1 \n", m1);
   /*
   matrix m1
   [[1.468433369525809,1.199383023707052,0.6154667729547869,2.830012157519816,-0.06995
    [2.638591175814763,2.966658948912904,2.001837257994767,-0.08570802469870731,0.8260
    [0.982665140643405,0.5825661973987162,-0.6615881071569371,0.4069533109215444,-0.16
    [-0.5791836119208447,-0.9332020478167575,2.801898707589577,-0.09887318478405316,-0
    [-0.1069569017521471,-0.04576879345014895,0.6320100840255642,-0.3805729270401813,0
   */

   matrix m2(3, 3);
   m2.Random(-2, 2);
   Print("matrix m2 \n", m2);
   /*
   matrix m2
   [[-1.928191291151613,-1.222873445419332,-1.965333905062949]
    [-1.077723600686929,1.053317863273741,-0.1786824246353196]
    [-1.965241401448204,1.620956906029246,-1.696386492740453]]
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量操纵

这些是基本矩阵操作的方法：填充、复制、获取矩阵的一部分、转置、拆分和排序。

还有若干种方法可针对矩阵的行和列进行操纵。

| 函数 | 动作 |
| --- | --- |
| HasNan | 返回矩阵/向量中NaN值的数量 |
| Transpose | 反转或置换矩阵的轴；返回修改后的矩阵 |
| TriL | 返回矩阵的副本，其中第 k 个对角线以上的元素归零。 下三角矩阵 |
| TriU | 返回矩阵的副本，其中第 k 对角线以下的元素归零。 上三角矩阵 |
| Diag | 提取对角线或构造对角矩阵 |
| Row | 返回行向量。写入向量指定行 |
| Col | 返回列向量。写入向量指定列。 |
| Copy | 返回给定矩阵/向量的副本 |
| Concat | 将两个子矩阵连接到一个矩阵。将两个向量连接到一个向量。 |
| Compare | 以指定的精度比较两个矩阵/向量的元素 |
| CompareByDigits | 比较两个矩阵/向量的元素，直至有效位数 |
| Flat | 允许依据一个索引替代两个索引定位矩阵元素 |
| Clip | 将矩阵/向量的元素限制为指定范围内的有效值 |
| Reshape | 在不更改矩阵数据的情况下更改矩阵的形状 |
| Resize | 返回形状和大小均已更改的新矩阵 |
| SwapRows | 交换矩阵中的行 |
| SwapCols | 交换矩阵中的列 |
| Split | 将矩阵拆分为多个子矩阵 |
| Hsplit | 将矩阵水平拆分为多个子矩阵。与 轴=0 的 Split 相同 |
| Vsplit | 将矩阵垂直拆分为多个子矩阵。与 axis=1 的拆分相同 |
| ArgSort | 间接针对矩阵或向量进行排序。 |
| Sort | 就地针对矩阵或向量进行排序。 |

© 2000-2025, MetaQuotes Ltd.

# HasNan

返回矩阵/向量中NaN值的数量。

```
ulong vector::HasNan();

ulong matrix::HasNan();
```

返回值

包含NaN值的矩阵/向量元素数。

注意

当比较具有NaN值的相应元素对时，Compare和CompareByDigits方法认为这些元素相等，而在通常比较浮点数NaN != NaN的情况下。

例如：

```
void OnStart(void)
  {
   double x=sqrt(-1);

   Print("single: ",x==x);

   vector<double> v1={x};
   vector<double> v2={x};

   Print("vector: ", v1.Compare(v2,0)==0);
  }

/* Result:

 single: false
 vector: true
*/
```

另见

MathClassify, Compare, CompareByDigits

© 2000-2025, MetaQuotes Ltd.

# Transpose

矩阵转置。反转或排列矩阵的轴；返回修改后的矩阵。

```
matrix matrix::Transpose()
```

返回值

转置后的矩阵。

以 MQL5 实现的简单矩阵转置算法：

```
matrix MatrixTranspose(const matrix& matrix_a)
  {
   matrix matrix_c(matrix_a.Cols(),matrix_a.Rows());

   for(ulong i=0; i<matrix_c.Rows(); i++)
      for(ulong j=0; j<matrix_c.Cols(); j++)
         matrix_c[i][j]=matrix_a[j][i];

   return(matrix_c);
  }
```

MQL5 示例：

```
  matrix a= {{0, 1, 2}, {3, 4, 5}};
  Print("matrix a \n", a);
  Print("a.Transpose() \n", a.Transpose());

  /*
  matrix a
  [[0,1,2]
   [3,4,5]]
  a.Transpose()
  [[0,3]
   [1,4]
   [2,5]]
  */
```

Python 示例：

```
import numpy as np

a = np.arange(6).reshape((2,3))
print("a \n",a)
print("np.transpose(a) \n",np.transpose(a))
```

© 2000-2025, MetaQuotes Ltd.

```
a
 [[0 1 2]
 [3 4 5]]
np.transpose(a)
 [[0 3]
 [1 4]
 [2 5]]
```

© 2000-2025, MetaQuotes Ltd.

# TransposeConjugate

Transposing a complex matrix with conjugation. Reverse or permute the axes of a matrix by changing a sign of an imaginary part of a complex number, return the modified matrix.

```
matrix matrix::TransposeConjugate()
```

### Return Value

Transposed complex conjugate matrix.

Simple algorithm of transposing a complex matrix with conjugation - method explanation

```
//--- Complex matrix transpose function with conjugation
matrix MatrixTranspose(const matrix& matrix_a) {
    //--- create a new matrix_c with dimensions inverse to matrix_a
    matrix matrix_c(matrix_a.Cols(), matrix_a.Rows());

    //--- go through all the rows of the new matrix
    for (ulong i = 0; i < matrix_c.Rows(); i++) {
        //--- go through all the columns of the new matrix
        for (ulong j = 0; j < matrix_c.Cols(); j++) {
            //--- transfer the real part of the element by transposing the index
            matrix_c[i][j].real = matrix_a[j][i].real;

            //--- transfer the imaginary part of the element by changing the sign (cor
            matrix_c[i][j].imag = -matrix_a[j][i].imag;
        }
    }

    //--- return the transposed matrix with conjugation
    return (matrix_c);
}
```

© 2000-2025, MetaQuotes Ltd.

# TriL

返回矩阵的副本，其中第 k 个对角线以上的元素归零。下三角矩阵。

```
matrix matrix::Tril(
  const int      ndiag=0       // 对角线的索引
   );
```

参数

*ndiag=0*

[输入] 对角线，高于该对角线的元素为零。ndiag = 0 (默认值)是主对角线，ndiag < 0 在其下方，ndiag > 0 位于其上方。

返回值

数组，其下三角以 1 填充，在其它所在则填充 0。

MQL5 示例：

```
   matrix a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   matrix b=a.TriL(-1);
   Print("matrix b \n",b);

  /*
  matrix_c
  [[0,0,0]
  [4,0,0]
  [7,8,0]
  [10,11,12]]
  */
```

Python 示例：

```
import numpy as np

a=np.tril([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)

[[ 0  0  0]
 [ 4  0  0]
 [ 7  8  0]
 [10 11 12]]
```

© 2000-2025, MetaQuotes Ltd.

# TriU

返回矩阵的副本，其中第 k 个对角线下方的元素归零。上三角形矩阵。

```
matrix matrix::Triu(
  const int      ndiag=0       // 对角线的索引
   );
```

参数

*ndiag=0*

[输入] 对角线之下的元素为零。ndiag = 0 (默认值)是主对角线，ndiag < 0 在其下方，ndiag > 0 位于其上方。

MQL5 示例：

```
   matrix a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   matrix b=a.TriU(-1);
   Print("matrix b \n",b);

  /*
  matrix b
  [[1,2,3]
   [4,5,6]
   [0,8,9]
   [0,0,12]]
  */
```

Python 示例：

```
import numpy as np

a=np.triu([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)
print(a)

[[ 1  2  3]
 [ 4  5  6]
 [ 0  8  9]
 [ 0  0 12]]
```

© 2000-2025, MetaQuotes Ltd.

# Diag

提取对角或构造对角矩阵。

```
vector matrix::Diag(
  const int      ndiag=0      // 对角线数
   );

void matrix::Diag(
  const vector   v,           // 对角线向量
  const int      ndiag=0      // 对角线数
   );
```

参数

*v*

[输入]一个向量，其元素将包含在相应的对角线中（ndiag=0 是主对角线）。

*ndiag=0*

[输入] 问题中的对角。默认填充值为 0。对于主对角线上方的对角线使用 ndiag>0，对于主对角线下方的对角线使用 ndiag<0。

注意

未分配的矩阵（没有维度）亦可设置对角线。在这种情况下，将创建一个大小为零矩阵，其大小对应于对角向量的大小，之后向量值将填充到相应的对角线中。 如果为已存在的矩阵设置对角线，则矩阵尺寸不会更改，且对角线向量外的矩阵元素值也不会更改。

举例

```
   vector v1={1,2,3};
   matrix m1;
   m1.Diag(v1);
   Print("m1\n",m1);
   matrix m2;
   m2.Diag(v1,-1);
   Print("m2\n",m2);
   matrix m3;
   m3.Diag(v1,1);
   Print("m3\n",m3);
   matrix m4=matrix::Full(4,5,9);
   m4.Diag(v1,1);
   Print("m4\n",m4);

   Print("diag -1 - ",m4.Diag(-1));
   Print("diag 0 - ",m4.Diag());
   Print("diag 1 - ",m4.Diag(1));
```

© 2000-2025, MetaQuotes Ltd.

```
/*

m1
[[1,0,0]
[0,2,0]
[0,0,3]]
m2
[[0,0,0]
[1,0,0]
[0,2,0]
[0,0,3]]
m3
[[0,1,0,0]
[0,0,2,0]
[0,0,0,3]]
m4
[[9,1,9,9,9]
[9,9,2,9,9]
[9,9,9,3,9]
[9,9,9,9,9]]
diag -1 - [9,9,9]
diag 0 - [9,9,9,9]
diag 1 - [1,2,3,9]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Row

返回行向量。写入向量指定行

```
vector matrix::Row(
  const ulong   nrow      // 行号
   );

void matrix::Row(
  const vector  v,        // 行向量
  const ulong   nrow      // 行号
   );
```

参数

*nrow*

[输入] 行号。

返回值

向量。

注意

可以为未分配的矩阵（没有维度）设置行。在这种情况下，将创建一个零矩阵，其大小为向量大小 x 行数+1，之后向量元素的值将填充在相应的行中。如果为已存在的矩阵设置行，则矩阵维度不会更改，行向量之外的矩阵元素的值也不会更改。

举例

```
   vector v1={1,2,3};
   matrix m1;
   m1.Row(v1,1);
   Print("m1\n",m1);
   matrix m2=matrix::Full(4,5,7);
   m2.Row(v1,2);
   Print("m2\n",m2);

   Print("row 1 - ",m2.Row(1));
   Print("row 2 - ",m2.Row(2));

  /*
  m1
  [[0,0,0]
  [1,2,3]]
  m2
  [[7,7,7,7,7]
```

© 2000-2025, MetaQuotes Ltd.

```
   [7,7,7,7,7]
   [1,2,3,7,7]
   [7,7,7,7,7]]
   row 1 - [7,7,7,7,7]
   row 2 - [1,2,3,7,7]
   */
```

© 2000-2025, MetaQuotes Ltd.

# Col

返回列向量。将向量写入指定的列。

```
vector matrix::Col(
  const ulong   ncol      // 列号
   );

void matrix::Col(
  const vector  v,        // 列向量
  const ulong   ncol      // 列号
   );
```

参数

*ncol*

[输入] 列数。

返回值

向量。

注意

可以为未分配的矩阵（没有维度）设置列数。在这种情况下，将创建一个零矩阵，其大小为向量大小 x 列数+1，之后向量元素的值将填充在相应的列中。如果为已存在的矩阵设置列，则矩阵维度不会变化，且列向量之外的矩阵元素值亦不会变化。

举例

```
   vector v1={1,2,3};
   matrix m1;
   m1.Col(v1,1);
   Print("m1\n",m1);
   matrix m2=matrix::Full(4,5,8);
   m2.Col(v1,2);
   Print("m2\n",m2);

   Print("col 1 - ",m2.Col(1));
   Print("col 2 - ",m2.Col(2));

  /*
  m1
  [[0,1]
  [0,2]
  [0,3]]
  m2
  [[8,8,1,8,8]
```

© 2000-2025, MetaQuotes Ltd.

```
   [8,8,2,8,8]
   [8,8,3,8,8]
   [8,8,8,8,8]]
   col 1 - [8,8,8,8]
   col 2 - [1,2,3,8]
   */
```

© 2000-2025, MetaQuotes Ltd.

# Copy

创建给定矩阵/向量的副本。

```
bool matrix::Copy(
  const matrix&   a      // 复制的矩阵
   );
bool vector::Copy(
  const vector&   v      // 复制的向量
   );
```

参数

*v*

[输入] 欲复制的矩阵或矢量。

返回值

成功时返回 true，否则返回 false。

MQL5 示例：

```
  matrix a=matrix::Eye(3, 4);
  matrix b;
  b.Copy(a);
  matrix c=a;
  Print("matrix b \n", b);
  Print("matrix_c \n", c);

  /*
  /*
  matrix b
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]]
  matrix_c
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]]
  */
  */
```

Python 示例：

```
import numpy as np

a = np.eye(3,4)
print('a \n',a)
b = a
```

© 2000-2025, MetaQuotes Ltd.

```
print('b \n',b)
c = np.copy(a)
print('c \n',c)

a
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
b
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
c
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

© 2000-2025, MetaQuotes Ltd.

# Concat

将两个子矩阵连接到一个矩阵。将两个向量连接到一个向量。

```
vector vector::Concat(
  const vector&  second_part      // 要连接的第二个向量
   );

vector matrix::Concat(
  const matrix&  second_part      // 要连接的第二个矩阵
   );

matrix matrix::Split(
  const matrix&  second_part,     // 要连接的第二个矩阵
  const int      axis             // axis
   );
```

参数

*second_part*

[in] 要连接的第二个向量或矩阵。如果矩阵沿着任何轴连接，那么矩阵的大小必须根据轴保持一致。

*axis*

[in] 轴。0-水平轴，1-垂直轴。

返回值

如果向量或矩阵在没有轴参数的情况下连接，或者矩阵在水平或垂直轴上连接，则返回向量。

示例

```
   vector vector_a={1,2,3,4};
   vector vector_b={5,6,7};
   vector vector_c=vector_a.Concat(vector_b);
   Print("vector_c=",vector_c);

   matrix matrix_a={{1,2,3},{4,5,6}};
   matrix matrix_b={{7,8,9},{10,11,12}};
   vector_c=matrix_a.Concat(matrix_b);
   Print("vector_c=",vector_c);

   matrix matrix_c0=matrix_a.Concat(matrix_b,0);
   Print("matrix_c0=\n",matrix_c0);

   matrix matrix_c1=matrix_a.Concat(matrix_b,1);
   Print("matrix_c1=\n",matrix_c1);

  /*
```

© 2000-2025, MetaQuotes Ltd.

```
vector_c=[1,2,3,4,5,6,7]
vector_c=[1,2,3,4,5,6,7,8,9,10,11,12]
matrix_c0=
[[1,2,3]
 [4,5,6]
 [7,8,9]
 [10,11,12]]
matrix_c1=
[[1,2,3,7,8,9]
 [4,5,6,10,11,12]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Compare

按指定的精度比较两个矩阵/向量的元素。

```
ulong vector::Compare(
  const vector& vec,          // 欲比较的向量
  const double  epsilon       // 精度
   );

ulong matrix::Compare(
  const matrix& mat,          // 欲比较的矩阵
  const double  epsilon       // 精度
   );
```

参数

*vector_b*

[输入] 欲比较的向量。

*epsilon*

[输入] 精度。

返回值

所比较的矩阵或向量的不匹配元素数量：如果矩阵相等，则为 0，否则大于 0。

注意

比较运算符 == 或 != 执行精确的元素级比较。众所周知，实数的精确比较用途有限，因此增加了 epsilon 比较方法。一个矩阵也许包含某个范围内的元素，例如从 1e-20 到 1e+20。这些矩阵可逐元素比较来进行处理，直到有效数字。

对于复数矩阵/向量，该比较包括估计复数之间的距离。距离的计算公式为sqrt(pow(r1-r2, 2) + pow(i1-i2, 2)，并且是一个可以与epsilon进行比较的实数。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4}};
   matrix matrix_i=matrix::Identity(3,3);
   matrix matrix_c=matrix_a.Inv();
   matrix matrix_check=matrix_a.MatMul(matrix_c);
   Print("matrix_check\n",matrix_check);

   ulong errors=matrix_check.Compare(matrix::Identity(3,3),1e-15);
   Print("errors=",errors);


  /*
  matrix_check
```

© 2000-2025, MetaQuotes Ltd.

```
   [[1,0,0]
   [4.440892098500626e-16,1,8.881784197001252e-16]
   [4.440892098500626e-16,2.220446049250313e-16,0.9999999999999996]]
   errors=0

   */
```

© 2000-2025, MetaQuotes Ltd.

# CompareByDigits

按有效位数精度比较两个矩阵/向量的元素。

```
ulong vector::CompareByDigits(
  const vector& vec,           // 欲比较的向量
  const int     digits         // 有效位数
   );

ulong matrix::CompareByDigits(
  const matrix& mat,           // 欲比较的矩阵
  const int     digits         // 有效位数
   );
```

参数

*vector_b*

[输入] 欲比较的向量。

*digits*

[输入] 进行比较采用的有效位数。

*epsilon*

[输入] 比较精度。如果两个值的绝对值相差小于指定的精度，则认为它们相等。

返回值

所比较的矩阵或向量的不匹配元素数量：如果矩阵相等，则为 0，否则大于 0。

注意

比较运算符 == 或 != 执行精确的元素级比较。众所周知，实数的精确比较用途有限，因此增加了 epsilon 比较方法。一个矩阵也许包含某个范围内的元素，例如从 1e-20 到 1e+20。此类矩阵可用元素级比较来处理，直至有效位数。

举例

```
   int    size_m=128;
   int    size_k=256;
   matrix matrix_a(size_m,size_k);
//--- 填充矩阵
   double  value=0.0;
   for(int i=0; i<size_m; i++)
     {
      for(int j=0; j<size_k; j++)
        {
         if(i==j)
            matrix_a[i][j]=1.0+i;
         else
```

© 2000-2025, MetaQuotes Ltd.

```
            {
             value+=1.0;
             matrix_a[i][j]=value/1e+20;
            }
         }
      }
//--- 获取其它矩阵
   matrix matrix_c = matrix_a * -1;

   ulong errors_epsilon=matrix_a.Compare(matrix_c,1e-15);
   ulong errors_digits=matrix_a.CompareByDigits(matrix_c,15);

   printf("Compare matrix %d x %d  errors_epsilon=%I64u  errors_digits=%I64u",size_m,s


  /*
  比较 矩阵 128 x 256  errors_epsilon=128  errors_digits=32768
  */
```

© 2000-2025, MetaQuotes Ltd.

# CompareEqual

Perform an absolute comparison of two matrices by unfolding successive rows into one-dimensional vectors.

```
ulong vector::Compare(
  const vector& vec    // vector to compare
   );

ulong matrix::CompareEqual(
  const matrix& mat     // matrix to compare
   );
```

### Parameters

*vec*

[in] Vector to compare.

*mat*

[in] Matrix to compare.

### Method description

Let us have two matrices: matrix A the method is called for and matrix B, which is passed as a method parameter. The comparison is performed as follows:

1. Matrices are expanded into one-dimensional vectors by successive concatenation of rows.
2. Vectors are compared element by element until the first mismatched element.
3. Depending on the comparison results, one of the values described below is returned.

### Return Value

-1 — if the matrix A element is less than the corresponding matrix B element.

0 — if all elements of A and B matrices are identical.

1 — if the matrix A element exceeds the corresponding matrix B element.

### Note

NaN value elements are taken into account when comparing.

NaN value elements are considered equal if they are present in both matrices at corresponding positions.

The NaN sign is not taken into account in the comparison.

An element with a NaN value is considered to be less than any other numeric value.

### Example

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

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   matrix matrix_a= {{10, 3, 2}, {1, 8, 12}, {6, 5, 4}};
   matrix matrix_i=matrix::Identity(3, 3);
   matrix matrix_c=matrix_a.Inv();
   matrix matrix_check=matrix_a.MatMul(matrix_c);
   Print("matrix_check\n", matrix_check);

   ulong errors=matrix_check.Compare(matrix::Identity(3, 3), 1e-15);
   Print("errors=", errors);
   /*
   matrix_check
   [[1,0,0]
   [4.440892098500626e-16,1,8.881784197001252e-16]
   [4.440892098500626e-16,2.220446049250313e-16,0.9999999999999996]]
   errors=0
   */

//---  absolute comparison of matrices
   matrix<double> A = matrix_a;  // Matrix A initialization
   matrix<double> B = matrix_c;  // Matrix B initialization
   int result = A.CompareEqual(B);
   switch(result)
     {
      case -1:
         Print("Matrix A is smaller than matrix B");
         break;
      case 0:
         Print("Matrices A and B are identical");
         break;
      case 1:
         Print("Matrix A is greater than matrix B");
         break;
      case -2:
         Print("Error! Matrix A is not initialized");
         break;
      case 2:
         Print("Error! Matrix B is not initialized");
         break;
      case -3:
         Print("Error! The size of matrix A is less than the size of matrix B");
         break;
      case 3:
         Print("Error! The size of matrix A is greater than the size of matrix B");
         break;
      default:
         Print("Error! Unknown error");
```

© 2000-2025, MetaQuotes Ltd.

```
            break;
        }
    }
```

© 2000-2025, MetaQuotes Ltd.

# Flat

允许依据一个索引替代两个索引来定位对矩阵元素。

```
bool matrix::Flat(
  const ulong   index,      //
  const double  value       // 设置的数值
   );

double matrix::Flat(
  const ulong   index,      //
   );
```

参数

*index*

[输入] 寻址索引

*value*

[输入] 给定索引值。

返回值

依据给定索引得到的值。

注意

对于矩阵 mat(3,3)，访问可以写成如下：

- 读取：'x=mat.Flat(4)'，其等效于 'x=mat[1][1]'
- 写入：'mat.Flat(5, 42)'，等效于 'mat[1][2]=42'

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);
   ulong arg_max=matrix_a.ArgMax();
   Print("max_value=",matrix_a.Flat(arg_max));
   matrix_a.Flat(arg_max,0);
   arg_max=matrix_a.ArgMax();
   Print("max_value=",matrix_a.Flat(arg_max));


  /*
  matrix_a
  [[10,3,2]
   [1,8,12]
   [6,5,4]
   [7,11,9]]
```

© 2000-2025, MetaQuotes Ltd.

```
   max_value=12.0
   max_value=11.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# Clip

将矩阵/向量的元素限制在指定的有效值范围内。

```
bool matrix::Clip(
  const double  min_value,      // 最小值
  const double  max_value       // 最大值
   );
bool vector::Clip(
  const double  min_value,      // 最小值
  const double  max_value       // 最大值
   );
```

参数

*min_value*

[输入] 最小值。

*max_value*

[输入] 最大值。

返回值

成功时返回 true，否则返回 false。

注意

矩阵（或向量）就地处理。不会创建任何副本。

举例

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   bool res=matrix_a.Clip(4,8);
   Print("matrix_a\n",matrix_a);

  /*
  matrix_a
  [[4,4,4]
   [4,5,6]
   [7,8,8]
   [8,8,8]]
  */
```

© 2000-2025, MetaQuotes Ltd.

# Reshape

在不更改其数据的情况下更改矩阵的形状。

```
void  Reshape(
  const ulong  rows,      // 新的行数。
  const ulong  cols       // 新的列数。
   );
```

参数

*rows*

[输入] 新的行数.

*cols*

[输入] 新的列数.

注意

矩阵就地处理。不会创建任何副本。可以指定任何大小，即，rows_new*cols_new!=rows_old*cols_old。当矩阵缓冲区增加时，额外数值均未定义。

举例

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

   Print("matrix_a\n",matrix_a);
   matrix_a.Reshape(2,6);
   Print("Reshape(2,6)\n",matrix_a);
   matrix_a.Reshape(3,5);
   Print("Reshape(3,5)\n",matrix_a);
   matrix_a.Reshape(2,4);
   Print("Reshape(2,4)\n",matrix_a);

  /*
  matrix_a
  [[1,2,3]
   [4,5,6]
   [7,8,9]
   [10,11,12]]
  Reshape(2,6)
  [[1,2,3,4,5,6]
   [7,8,9,10,11,12]]
  Reshape(3,5)
  [[1,2,3,4,5]
   [6,7,8,9,10]
   [11,12,0,3,0]]
  Reshape(2,4)
  [[1,2,3,4]
   [5,6,7,8]]
```

© 2000-2025, MetaQuotes Ltd.

```
*/
```

© 2000-2025, MetaQuotes Ltd.

# Resize

返回已更改形状和大小的新矩阵。

```
bool matrix::Resize(
  const ulong  rows,     //
  const ulong  cols,     // 新列数。
  const ulong  reserve=0 // 项目保留额度。
   );

bool vector::Resize(
  const ulong  size,     // 新大小。
  const ulong  reserve=0 // 项目保留额度。
   );
```

参数

*rows*

[输入] 新的行数.

*cols*

[输入] 新的列数.

返回值

成功时返回 true，否则返回 false。

注意

矩阵（或向量）就地处理。不会创建任何副本。可以指定任何大小，即，rows_new*cols_new!=rows_old*cols_old。与 Reshape 不同，矩阵是逐行处理的。当增加列数时，额外列的数值均未定义。当增加行数时，额外行的数值均未定义。当减少列数时，矩阵的每一行都会被截断。

举例

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);
   matrix_a.Resize(2,6);
   Print("Ressize(2,6)\n",matrix_a);
   matrix_a.Resize(3,5);
   Print("Resize(3,5)\n",matrix_a);
   matrix_a.Resize(2,4);
   Print("Resize(2,4)\n",matrix_a);

   /*
   matrix_a
   [[1,2,3]
    [4,5,6]
```

© 2000-2025, MetaQuotes Ltd.

```
    [7,8,9]
    [10,11,12]]
  Ressize(2,6)
  [[1,2,3,4,5,6]
    [4,5,6,10,11,12]]
  Resize(3,5)
  [[1,2,3,4,5]
    [4,5,6,10,11]
    [11,12,3,8,8]]
  Resize(2,4)
  [[1,2,3,4]
    [4,5,6,10]]
  */
```

© 2000-2025, MetaQuotes Ltd.

## Set

根据指定索引设置向量元素值。

```
bool vector::Set(
  ulong   index,      // 元素索引
  double  value       // 值
   );
```

参数

*index*

[in] 需要为其设置值的元素索引。

*值*

[in] 值。

返回值

如果成功返回true，否则返回false。

注意

Set方法的作用与使用方括号赋值的作用相同，即：*vector[index]=value*. 添加该方法是为了简化从使用此类符号的语言中传递代码的过程。下面的示例显示了按指定索引用值填充向量的两个选项。

示例：

```
void OnStart()
  {
//---
   vector v1(10, VectorAssignValues);
   Print("v1 = ", v1);

   vector v2(10, VectorSetValues);
   Print("v2 = ", v2);
  }
 /* Result
  v1 = [1,2,4,8,16,32,64,128,256,512]
  v2 = [1,2,4,8,16,32,64,128,256,512]
  */
//+------------------------------------------------------------------------+
//| 通过赋值运算用数字的幂填充向量                                         |
//+------------------------------------------------------------------------+
void VectorAssignValues(vector& v, double initial=1)
  {
   double value=initial;
   for(ulong k=0; k<v.Size(); k++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       v[k]=value;
       value*=2;
      }
   }
//+------------------------------------------------------------------------+
//| 使用Set方法用数字的幂填充向量                                          |
//+------------------------------------------------------------------------+
void VectorSetValues(vector& v, double initial=1)
   {
    double value=initial;
    for(ulong k=0; k<v.Size(); k++)
      {
       v.Set(k, value);
       value*=2;
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## SwapRows

交换矩阵中的行。

```
bool matrix::SwapRows(
  const ulong  row1,     // 第一行的索引
  const ulong  row2      // 第二行的索引
   );
```

参数

*row1*

[输入] 第一个行的索引。

*row2*

[输入] 第二个行的索引。

返回值

成功时返回 true，否则返回 false。

举例

```
   matrix matrix_a={{1,2,3,4},
                    {5,6,7,8},
                    {9,10,11,12},
                    {13,14,15,16}};
   matrix matrix_i=matrix::Identity(4,4);
   matrix matrix_a1=matrix_a;
   matrix_a1.SwapRows(0,3);
   Print("matrix_a1\n",matrix_a1);

   matrix matrix_p=matrix_i;
   matrix_p.SwapRows(0,3);
   matrix matrix_c1=matrix_p.MatMul(matrix_a);
   Print("matrix_c1\n",matrix_c1);

  /*
  matrix_a1
  [[13,14,15,16]
   [5,6,7,8]
   [9,10,11,12]
   [1,2,3,4]]
  matrix_c1
  [[13,14,15,16]
   [5,6,7,8]
   [9,10,11,12]
   [1,2,3,4]]
  */
```

© 2000-2025, MetaQuotes Ltd.

## SwapCols

交换矩阵中的列。

```
bool matrix::SwapCols(
  const ulong  row1,      // 第一列的索引
  const ulong  row2       // 第二列的索引
   );
```

参数

*col1*

[输入] 第一个列的索引。

*col2*

[输入] 第二个列的索引。

返回值

成功时返回 true，否则返回 false。

举例

```
   matrix matrix_a={{1,2,3,4},
                    {5,6,7,8},
                    {9,10,11,12},
                    {13,14,15,16}};
   matrix matrix_i=matrix::Identity(4,4);
   matrix matrix_a1=matrix_a;
   matrix_a1.SwapCols(0,3);
   Print("matrix_a1\n",matrix_a1);

   matrix matrix_p=matrix_i;
   matrix_p.SwapCols(0,3);
   matrix matrix_c1=matrix_a.MatMul(matrix_p);
   Print("matrix_c1\n",matrix_c1);

  /*
  matrix_a1
  [[4,2,3,1]
   [8,6,7,5]
   [12,10,11,9]
   [16,14,15,13]]
  matrix_c1
  [[4,2,3,1]
   [8,6,7,5]
   [12,10,11,9]
   [16,14,15,13]]
  */
```

© 2000-2025, MetaQuotes Ltd.

# Split

将矩阵拆分为多个子矩阵。

```
bool matrix::Split(
  const ulong  parts,      // 子矩阵数
  const int    axis,       // 轴
  matrix&      splitted[]  // 结果子矩阵数组
   );

void matrix::Split(
  const ulong& parts[],    // 子矩阵大小
  const int    axis,       // 轴
  matrix&      splitted[]  // 结果子矩阵数组
   );
```

参数

*parts*

[输入]欲将矩阵划分为子矩阵的数量。

*axis*

[输入] 轴。0 - 水平轴，1 - 垂直轴。

*splitted*

[输出]生成的子矩阵数组。

返回值

成功时返回 true，否则返回 false。

注意

如果指定了子矩阵的数量，则得到的子矩阵相同大小。这意味着矩阵大小 (0 - 行数，1 - 列数)必须能被 "parts" 参数整除，没有余数。也可按子矩阵尺寸数组得到不同尺寸的子矩阵。各尺寸数组的元素都可用，直到整个矩阵被分割。如果数组尺寸分割结束，而矩阵尚有部分未除尽，则未分割的余数将作为最后一个子矩阵。

举例

```
   matrix matrix_a={{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9,10,11,12},
                    {13,14,15,16,17,18},
                    {19,20,21,22,23,24},
                    {25,26,27,28,29,30}};
   matrix splitted[];
   ulong  parts[]={2,2};

   bool res=matrix_a.Split(2,0,splitted);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print(res,"  ",GetLastError());
   ResetLastError();
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   res=matrix_a.Split(2,1,splitted);
   Print(res,"  ",GetLastError());
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   res=matrix_a.Split(parts,0,splitted);
   Print(res,"  ",GetLastError());
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

  /*
  false  4003
  true  0
  splitted 0
  [[1,2,3]
   [7,8,9]
   [13,14,15]
   [19,20,21]
   [25,26,27]]
  splitted 1
  [[4,5,6]
   [10,11,12]
   [16,17,18]
   [22,23,24]
   [28,29,30]]
  true  0
  splitted 0
  [[1,2,3,4,5,6]
   [7,8,9,10,11,12]]
  splitted 1
  [[13,14,15,16,17,18]
   [19,20,21,22,23,24]]
  splitted 2
  [[25,26,27,28,29,30]]
  */
```

© 2000-2025, MetaQuotes Ltd.

# Hsplit

将矩阵水平拆分为多个子矩阵。与 axis=0 的拆分相同

```
bool matrix::Hsplit(
  const ulong  parts,      // 子矩阵数
  matrix&      splitted[]  // 结果子矩阵数组
   );

void matrix::Hsplit(
  const ulong& parts[],    // 子矩阵大小
  matrix&      splitted[]  // 结果子矩阵数组
   );
```

参数

*parts*

[输入] 欲将矩阵划分为子矩阵的数量。

*splitted*

[输出] 生成的子矩阵数组。

返回值

成功时返回 true，否则返回 false。

注意

如果指定了子矩阵的数量，则得到的子矩阵相同大小。这意味着行数必须能被 "parts" 参数整除，没有余数。也可按子矩阵尺寸数组得到不同尺寸的子矩阵。各尺寸数组的元素都可用，直到整个矩阵被分割。 如果数组尺寸分割结束，而矩阵尚有部分未除尽，则未分割的余数将作为最后一个子矩阵。

举例

```
   matrix matrix_a={{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9,10,11,12},
                    {13,14,15,16,17,18},
                    {19,20,21,22,23,24},
                    {25,26,27,28,29,30}};
   matrix splitted[];
   ulong  parts[]={2,4};

   bool res=matrix_a.Hsplit(2,splitted);
   Print(res,"  ",GetLastError());
   ResetLastError();
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);
```

© 2000-2025, MetaQuotes Ltd.

```
   res=matrix_a.Hsplit(5,splitted);
   Print(res,"  ",GetLastError());
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   res=matrix_a.Hsplit(parts,splitted);
   Print(res,"  ",GetLastError());
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);


/*
false  4003
true  0
splitted 0
[[1,2,3,4,5,6]]
splitted 1
[[7,8,9,10,11,12]]
splitted 2
[[13,14,15,16,17,18]]
splitted 3
[[19,20,21,22,23,24]]
splitted 4
[[25,26,27,28,29,30]]
true  0
splitted 0
[[1,2,3,4,5,6]
[7,8,9,10,11,12]]
splitted 1
[[13,14,15,16,17,18]
[19,20,21,22,23,24]
[25,26,27,28,29,30]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Vsplit

将矩阵垂直拆分为多个子矩阵。axis=1 的拆分相同

```
bool matrix::Vsplit(
  const ulong  parts,      // 子矩阵数
  matrix&      splitted[]  // 结果子矩阵数组
   );

void matrix::Vsplit(
  const ulong& parts[],    // 子矩阵大小
  matrix&      splitted[]  // 结果子矩阵数组
   );
```

参数

*parts*

[输入] 欲将矩阵划分为子矩阵的数量。

*splitted*

[输出] 生成的子矩阵数组。

返回值

成功时返回 true，否则返回 false。

注意

如果指定了子矩阵的数量，则得到的子矩阵相同大小。这意味着列数必须能被 "parts" 参数整除，没有余数。也可按子矩阵尺寸数组得到不同尺寸的子矩阵。各尺寸数组的元素都可用，直到整个矩阵被分割。 如果数组尺寸分割结束，而矩阵尚有部分未除尽，则未分割的余数将作为最后一个子矩阵。

举例

```
   matrix matrix_a={{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9,10,11,12},
                    {13,14,15,16,17,18}};
   matrix splitted[];
   ulong  parts[]={2,3};

   matrix_a.Vsplit(2,splitted);
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   matrix_a.Vsplit(3,splitted);
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   matrix_a.Vsplit(parts,splitted);
```

© 2000-2025, MetaQuotes Ltd.

```
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);


  /*
     splitted 0
     [[1,2,3]
      [7,8,9]
      [13,14,15]]
     splitted 1
     [[4,5,6]
      [10,11,12]
      [16,17,18]]

     splitted 0
     [[1,2]
      [7,8]
      [13,14]]
     splitted 1
     [[3,4]
      [9,10]
      [15,16]]
     splitted 2
     [[5,6]
      [11,12]
      [17,18]]

     splitted 0
     [[1,2]
      [7,8]
      [13,14]]
     splitted 1
     [[3,4,5]
      [9,10,11]
      [15,16,17]]
     splitted 2
     [[6]
      [12]
      [18]]

  */
```

© 2000-2025, MetaQuotes Ltd.

# ArgSort

矩阵或向量的间接排序。

```
vector vector::Sort(
  func_name  compare_func=NULL,  // 比较函数
  T          context             // 自定义排序函数的参数
   );

matrix matrix::Sort(
  func_name  compare_func=NULL   // 比较函数
  T          context             // 自定义排序函数的参数
   );

matrix matrix::Sort(
  const int  axis,               // 排序轴
  func_name  compare_func=NULL   // 比较函数
  T          context             // 自定义排序函数的参数
   );
```

参数

*axis*

[输入] 要排序的轴：0 是水平，1 是垂直。

*func_name*

[输入] 比较器。您可以指定 ENUM_SORT_MODE 枚举值之一，或您自己的比较函数。如果未指定函数，则采用升序排序。

自定义比较函数可以有两种类型：

- int comparator(T x1,T x2)
- int comparator(T x1,T x2,TContext context)

此处 T 是矩阵或向量的类型，而传递给 Sort 方法的附加参数 TContex 则是作为 "上下文关联"变量类型。

*context*

[输入] 可传递给自定义排序函数的其它可选参数。

返回值

含有已排序元素索引的向量或矩阵。例如，结果 [4,2,0,1,3] 指示元素 4 其索引应该位于零位置，元素 2 其索引位于第一个位置，依此类推。

© 2000-2025, MetaQuotes Ltd.

## Sort

就地针对矩阵或向量进行排序。

```
void vector::Sort(
  func_name  compare_func=NULL,  // 比较函数
  T          context             // 自定义排序函数的参数
   );

void matrix::Sort(
  func_name  compare_func=NULL   // 比较函数
  T          context             // 自定义排序函数的参数
   );

void matrix::Sort(
  const int  axis,               // 排序轴
  func_name  compare_func=NULL   // 比较函数
  T          context             // 自定义排序函数的参数
   );
```

参数

*axis*

[输入] 要排序的轴：0 是水平，1 是垂直。

*func_name*

[输入] 比较器。您可以指定 ENUM_SORT_MODE 枚举值之一，或您自己的比较函数。如果未指定函数，则采用升序排序。

自定义比较函数可以有两种类型：

- int comparator(T x1,T x2)
- int comparator(T x1,T x2,TContex context)

此处 T 是矩阵或向量的类型，而传递给 Sort 方法的附加参数 TContex 则是作为 "上下文关联"变量类型。

*context*

[输入] 可传递给自定义排序函数的其它可选参数。

返回值

无。排序就地执行，即它应用于调用 Sort 方法的矩阵/向量的数据。

举例

```
//+------------------------------------------------------------------+
//| Sort 函数                                                        |
//+------------------------------------------------------------------+
int MyDoubleComparator(double x1,double x2,int sort_mode=0)
  {
   int res=x1<x2 ? -1 : (x1>x2 ? 1 : 0);
```

© 2000-2025, MetaQuotes Ltd.

```
    return(sort_mode==0 ? res : -res);
   }
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
   {
    //--- 填充向量
    vector v(100);
    //--- 升序排序
    v.Sort(MyDoubleComparator);    // 此处使用了默认值为 “0” 的附加参数
    Print(v);
    // 降序排序
    v.Sort(MyDoubleComparator,1); // 此处的附加参数 “1” 是由用户显式指定的
    Print(v);
   }
```

© 2000-2025, MetaQuotes Ltd.

# 矩阵和向量的数学运算

数学运算，包括加法、减法、乘法和除法，可以针对矩阵和向量按元素级执行。

数学函数最初设计用于对标量值执行相关操作。大多数函数可以应用于矩阵和向量。其中包括 MathAbs, MathArccos, MathArcsin, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathLog10, MathMod, MathPow, MathRound, MathSin, MathSqrt, MathTan, MathExpm1, MathLog1p, MathArccosh, MathArcsinh, MathArctanh, MathCosh, MathSinh, 和 MathTanh。此类操作意味着矩阵和向量的元素级处理。示例

```
//---
  matrix a= {{1, 4}, {9, 16}};
  Print("matrix a=\n",a);
  a=MathSqrt(a);
  Print("MatrSqrt(a)=\n",a);
  /*
   matrix a=
    [[1,4]
     [9,16]]
   MatrSqrt(a)=
    [[1,2]
     [3,4]]
  */
```

对于 MathMod 和 MathPow，第二个元素可以是标量，，或相应大小的矩阵/向量。

下面的示例展示如何将数学函数应用于向量来计算标准偏差。

```
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- 使用初始化函数填充矢量
  vector r(10, ArrayRandom); // 从0 到1 的随机数数组
//--- 计算平均值
  double avr=r.Mean();       // 数组平均值
  vector d=r-avr;            // 计算与平均值的偏差数组
  Print("avr(r)=", avr);
  Print("r=", r);
  Print("d=", d);
  vector s2=MathPow(d, 2);   // 方差平方数组
  double sum=s2.Sum();       // 方差平方和
//--- 以两种不同的方法计算标准偏差
  double std=MathSqrt(sum/r.Size());
  Print(" std(r)=", std);
  Print("r.Std()=", r.Std());
 }
/*
  avr(r)=0.5300302133243813
```

© 2000-2025, MetaQuotes Ltd.

```
   r=[0.8346201971495713,0.8031556138798182,0.6696676534318063,0.05386516922513505,0.54
   d=[0.30458998382519,0.2731254005554369,0.1396374401074251,-0.4761650440992462,0.0190
    std(r)=0.2838269732183663
   r.Std()=0.2838269732183663
*/
//+------------------------------------------------------------------+
//| 以随机数值填充向量                                               |
//+------------------------------------------------------------------+
void ArrayRandom(vector& v)
 {
  for(ulong i=0; i<v.Size(); i++)
    v[i]=double(MathRand())/32767.;
 }
```

© 2000-2025, MetaQuotes Ltd.

# 数学运算

数学运算，包括加法、减法、乘法和除法，可以针对矩阵和向量按元素级执行。

两个矩阵或两个向量必须属于同一类型，并且必须具有相同的维度。矩阵的每个元素都对第二个矩阵的相应元素进行操作。

您还可以使用相应的类型（双精度、浮点数或复数）的标量作为第二项（乘数、减法或除数）。在这种情况下，矩阵或向量的每个成员都将在指定的标量上运行。

```
   matrix matrix_a={{0.1,0.2,0.3},{0.4,0.5,0.6}};
   matrix matrix_b={{1,2,3},{4,5,6}};

   matrix matrix_c1=matrix_a+matrix_b;
   matrix matrix_c2=matrix_b-matrix_a;
   matrix matrix_c3=matrix_a*matrix_b;    // 哈达玛（Hadamard）乘积，不要和矩阵乘积混淆！ 对此有
   matrix matrix_c4=matrix_b/matrix_a;

   matrix_c1=matrix_a+1;
   matrix_c2=matrix_b-double_value;
   matrix_c3=matrix_a*M_PI;
   matrix_c4=matrix_b/0.1;

//--- 可以就地采取行动
   matrix_a+=matrix_b;
   matrix_a/=2;
```

相同的操作可用于向量。

© 2000-2025, MetaQuotes Ltd.

# 数学函数

以下数学函数可以应用于矩阵和向量：MathAbs, MathArccos, MathArcsin, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathLog10, MathMod, MathPow, MathRound, MathSin, MathSqrt, MathTan, MathExpm1, MathLog1p, MathArccosh, MathArcsinh, MathArctanh, MathCosh, MathSinh, MathTanh。此类操作意味着矩阵和向量的元素级处理。

对于 MathMod 和 MathPow，第二个元素既可以是标量，亦或相应大小的矩阵/向量。

```
   matrix<T> mat1(128,128);
   matrix<T> mat3(mat1.Rows(),mat1.Cols());
   ulong     n,size=mat1.Rows()*mat1.Cols();
...
   mat2=MathPow(mat1,(T)1.9);
   for(n=0; n<size; n++)
     {
      T res=MathPow(mat1.Flat(n),(T)1.9);
      if(res!=mat2.Flat(n))
         errors++;
     }
   mat2=MathPow(mat1,mat3);
   for(n=0; n<size; n++)
     {
      T res=MathPow(mat1.Flat(n),mat3.Flat(n));
      if(res!=mat2.Flat(n))
         errors++;
     }
...
   vector<T> vec1(16384);
   vector<T> vec3(vec1.Size());
   ulong     n,size=vec1.Size();
...
   vec2=MathPow(vec1,(T)1.9);
   for(n=0; n<size; n++)
     {
      T res=MathPow(vec1[n],(T)1.9);
      if(res!=vec2[n])
         errors++;
     }
   vec2=MathPow(vec1,vec3);
   for(n=0; n<size; n++)
     {
      T res=MathPow(vec1[n],vec3[n]);
      if(res!=vec2[n])
         errors++;
     }
```

© 2000-2025, MetaQuotes Ltd.

# 矩阵和矢量乘积

矩阵和向量乘积计算包括：

- 矩阵乘法
- 向量乘法
- 计算协方差矩阵
- 计算两个向量的互相关
- 计算两个向量的卷积
- 计算相关系数

| 函数 | 动作 |
| --- | --- |
| MatMul | 两个矩阵的矩阵乘积 |
| GeMM | 通用矩阵乘法（GeMM） |
| Power | 将方阵提升整数幂 |
| Dot | 两个向量的点积 |
| Kron | 返回两个矩阵、矩阵和向量，向量和矩阵、或两个向量的克罗内克（Kronecker)乘积 |
| Inner | 两个矩阵的内积 |
| Outer | 计算两个矩阵或两个向量的外积 |
| CorrCoef | 计算皮尔逊（Pearson)相关系数（线性相关系数） |
| Cov | 计算协方差矩阵 |
| Correlate | 计算两个向量的互相关性 |
| Convolve | 返回两个向量的离散线性卷积 |

© 2000-2025, MetaQuotes Ltd.

# MatMul

MatMul方法支持矩阵和向量的乘法，具有若干个重载。

矩阵乘以矩阵：matrix[M][K] * matrix[K][N] = matrix[M][N]

```
matrix  matrix::MatMul(
  const matrix&  b      // 第二个矩阵
   );
```

向量乘以矩阵：horizontal vector[K] * matrix[K][N] = horizontal vector[N]

```
vector  vector::MatMul(
  const matrix&  b      // 矩阵
   );
```

矩阵乘以向量：matrix[M][K] * vertical vector[K] = vertical vector[M]

```
vector  matrix::MatMul(
  const vector&  b      // 向量
   );
```

标量向量乘法：horizontal vector * vertical vector = dot value

```
scalar  vector::MatMul(
  const vector&  b      // 第二个向量
   );
```

参数

*b*

[输入] 矩阵或向量。

返回值

矩阵、矢量或标量，具体取决于所使用的方法。

注意

矩阵应兼容乘法，即第一个矩阵中的列数应等于第二个矩阵中的行数。 矩阵乘法不支持交换律：在一般情况下，将第一个矩阵乘以第二个矩阵的结果不等于将第二个矩阵乘以第一个矩阵的结果。

矩阵积由第一个矩阵的行向量和第二个矩阵的列向量的标量积的所有可能组合组成。

在标量乘法中，向量必须具有相同的长度。

当向量和矩阵相乘时，向量的长度必须与矩阵中的列数完全匹配。

以MQL5实现的朴素矩阵乘法算法：

```
matrix MatrixProduct(const matrix& matrix_a, const matrix& matrix_b)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   matrix matrix_c;

   if(matrix_a.Cols()!=matrix_b.Rows())
      return(matrix_c);

   ulong M=matrix_a.Rows();
   ulong K=matrix_a.Cols();
   ulong N=matrix_b.Cols();
   matrix_c=matrix::Zeros(M,N);

   for(ulong m=0; m<M; m++)
      for(ulong k=0; k<K; k++)
         for(ulong n=0; n<N; n++)
            matrix_c[m][n]+=matrix_a[m][k]*matrix_b[k][n];

   return(matrix_c);
  }
```

矩阵乘法示例

```
   matrix a={{1, 0, 0},
             {0, 1, 0}};
   matrix b={{4, 1},
             {2, 2},
             {1, 3}};
   matrix c1=a.MatMul(b);
   matrix c2=b.MatMul(a);
   Print("c1 = \n", c1);
   Print("c2 = \n", c2);
/*
   c1 =
   [[4,1]
    [2,2]]
   c2 =
   [[4,1,0]
    [2,2,0]
    [1,3,0]]
*/
```

水平向量乘以矩阵的示例

```
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建一个 3x5 矩阵
   matrix m35;
   m35.Init(3, 5, Arange);
//---
   vector v3 = {1, 2, 3};
   Print("Product of horizontal vector v and matrix m[3,5]");
   Print("On the left, vector v3 = ", v3);
   Print("On the right, matrix m35 = \n", m35);
   Print("v3.MatMul(m35) = horizontal vector v[5] \n", v3.MatMul(m35));

  /* 结果
    Product of horizontal vector v3 and matrix m[3,5]
    On the left, vector v3 = [1,2,3]
    On the right, matrix m35 =
    [[0,1,2,3,4]
     [5,6,7,8,9]
     [10,11,12,13,14]]
    v3.MatMul(m35) = horizontal vector v[5]
    [40,46,52,58,64]
   */
  }
//+------------------------------------------------------------------+
//|  以递增数值填充矩阵                                              |
//+------------------------------------------------------------------+
void Arange(matrix & m, double start = 0, double step = 1)
  {
//---
   ulong cols = m.Cols();
   ulong rows = m.Rows();
   double value = start;
   for(ulong r = 0; r < rows; r++)
     {
      for(ulong c = 0; c < cols; c++)
        {
         m[r][c] = value;
         value += step;
        }
     }
//---
  }
```

矩阵如何乘以垂直向量的示例

```
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建一个 3x5 矩阵
   matrix m35;
   m35.Init(3, 5, Arange);
//---
   Print("Product of matrix m[3,5] and vertical vector v[5]");
   vector v5 = {1,2,3,4,5};
   Print("On the left, m35 = \n",m35);
   Print("On the right v5 = ",v5);
   Print("m35.MatMul(v5) = vertical vector v[3] \n",m35.MatMul(v5));

  /* 结果
   Product of matrix m[3,5] and vertical vector v[5]
   On the left, m35 =
   [[0,1,2,3,4]
    [5,6,7,8,9]
    [10,11,12,13,14]]
   On the right, v5 = [1,2,3,4,5]
   m35.MatMul(v5) = vertical vector v[3]
   [40,115,190]
   */
  }
//+------------------------------------------------------------------+
//|   以递增数值填充矩阵                                             |
//+------------------------------------------------------------------+
void Arange(matrix & m, double start = 0, double step = 1)
  {
//---
   ulong cols = m.Cols();
   ulong rows = m.Rows();
   double value = start;
   for(ulong r = 0; r < rows; r++)
     {
      for(ulong c = 0; c < cols; c++)
        {
         m[r][c] = value;
         value += step;
        }
     }
//---
  }
```

向量的标量（点）积示例

```
void OnStart()
  {
//--- 水平向量和垂直向量的标量乘积
   vector a= {1, 2, 3};  // 水平向量
   vector b= {4, 5, 6};  // 垂直向量
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("a = ", a);
   Print("b = ", b);
   Print("1) a.MatMul(b) = ", a.MatMul(b));
   //--- 参见 Dot 方法生成相同的结果
   Print("2) a.Dot(b) = ", a.Dot(b));

  /* 结果
   a = [1,2,3]
   b = [4,5,6]
   1) a.MatMul(b) = 32.0
   2) a.Dot(b) = 32.0
    */
  }
```

参见

Dot, GeMM

© 2000-2025, MetaQuotes Ltd.

## GeMM

GeMM 通用矩阵乘法)方法实现了两个矩阵的一般乘法。运算定义为 C ← αA B + βC，其中 A 和 B 矩阵可以选择转置。配合矩阵 AB (MatMul)的正态乘法，假定 *alpha* 标量等于 1，*beta* 等于零。

GeMM 和 MatMul 在效率方面的主要区别在于，MatMul 总是创建一个新的矩阵/矢量对象，而 GeMM 使用现有的矩阵对象，且不会重新创建它。因此，当您使用 GeMM 并为相应的矩阵预分配内存时，在操控相同的矩阵大小时，将不会重新分配内存。这可能是 GeMM 在批量计算方面的一个重要优势，例如，在策略测试器中运行优化、或训练神经网络时。

与 MatMul类似，GeMM 也有 4 个重载。然而，第四次重载的语义已被修改，以便能够将垂直向量与水平向量相乘。

在现有的矩阵/矢量对象中，没有必要预先分配内存。将在第一次调用 GeMM 时分配内存，并以零填充。

矩阵乘以矩阵： `matrix C[M][N] = α*(matrix A[M][K] * matrix B[K][N]) + β* matrix C[M][N]`

```
bool  matrix::GeMM(
  const matrix &A,     // first matrix
  const matrix &B,     // second matrix
  double alpha,        // 乘积 AB 的 alpha 乘数
  double beta,         // beta multiplier for matrix C
  uint   flags         // a combination of ENUM_GEMM values (bitwise OR), which determi
   );
```

向量乘以矩阵： `vector C[N] = α*(vector A[K] * matrix B[K][N]) + β* vector C[N]`

```
bool  vector::GeMM(
  const vector &A,     // 水平向量
  const matrix &B,     // 矩阵
  double alpha,        // 乘积 AB 的 alpha 乘数
  double beta,         // 向量 C 的 beta 乘数
  uint   flags         // ENUM_GEMM 枚举值，判定是否矩阵 A 是转置
   );
```

矩阵乘以向量： `vector C[M] = α*( matrix A[M][K] *vector B[K] * ) + β* vector C[M]`

```
bool  vector::GeMM(
  const matrix &A,     // 矩阵
  const vector &B,     // 垂直向量
  double alpha,        // 乘积 AB 的 alpha 乘数
  double beta,         // 向量 C 的 beta 乘数
  uint   flags         // ENUM_GEMM 枚举值，判定是否矩阵 B 是转置
   );
```

向量乘以向量： `matrix C[M][N] = α*( vector A[M] *vector B[N] * ) + β* matrix C[M][N]`. 此重载返回一个矩阵，与 MatMul不同，它返回一个标量。

```
bool  matrix::GeMM(
  const vector &A,     // 第一个向量
```

© 2000-2025, MetaQuotes Ltd.

```
   const vector &B,     // 第二个向量
   double alpha,        // 乘积 AB 的 alpha 乘数
   double beta,         // 用于矩阵 C 的 beta
   uint   flags         // ENUM_GEMM 枚举值，判定是否矩阵 C 是转置
    );
```

参数

*A*

[输入] 矩阵或向量。

*B*

[输入] 矩阵或向量。

*alpha*

[输入] AB 乘积的阿尔法乘数。

*beta*

[输入] 所得 C 矩阵的贝塔乘数。

*flags*

[输入] 确定是否转置矩阵 A、B 和 C 的 ENUM_GEMM 枚举值。

返回值

成功返回 true，否则 false。

**ENUM_GEMM**

GeMM 方法的标志枚举。

| ID | 说明 |
| --- | --- |
| TRANSP_A | 使用转置矩阵 A |
| TRANSP_B | 使用转置矩阵 B |
| TRANSP_C | 使用转置矩阵 C |

注意

浮点型、双精度型和复数类型的矩阵和向量可以用作参数 A 和 B。GeMM 方法的模板变体如下：

```
bool matrix<T>::GeMM(const matrix<T> &A,const matrix<T> &B,T alpha,T beta,ulong flags)
bool matrix<T>::GeMM(const vector<T> &A,const vector<T> &B,T alpha,T beta,ulong flags)

bool vector<T>::GeMM(const vector<T> &A,const matrix<T> &B,T alpha,T beta,ulong flags)
bool vector<T>::GeMM(const matrix<T> &A,const vector<T> &B,T alpha,T beta,ulong flags)
```

© 2000-2025, MetaQuotes Ltd.

基本上一般的矩阵乘法函数描述为：

```
C[m,n] = α*Sum(A[m,k]*B[k,n]) + β*C[m,n]
```

配合以下尺寸：矩阵 A 为 M x K，矩阵 B 为 K x N，矩阵 C 为 M x N。

因此，矩阵应该与乘法兼容，即第一个矩阵中的列数应等于第二个矩阵中的行数。矩阵乘法不支持交换律：在一般情况下，将第一个矩阵乘以第二个矩阵的结果不等于将第二个矩阵乘以第一个矩阵的结果。

示例：

```
void OnStart()
  {
   vector vector_a= {1, 2, 3, 4, 5};
   vector vector_b= {4, 3, 2, 1};
   matrix matrix_c;
//--- 计算两个向量的 GeMM
   matrix_c.GeMM(vector_a, vector_b, 1, 0);
   Print("matrix_c:\n ", matrix_c, "\n");
   /*
   matrix_c:
    [[4,3,2,1]
    [8,6,4,2]
    [12,9,6,3]
    [16,12,8,4]
    [20,15,10,5]]
   */
//--- 创建矩阵作为向量
   matrix matrix_a(5, 1);
   matrix matrix_b(1, 4);
   matrix_a.Col(vector_a, 0);
   matrix_b.Row(vector_b, 0);
   Print("matrix_a:\n ", matrix_a);
   Print("matrix_b:\n ", matrix_b);
   /*
   matrix_a:
   [[1]
   [2]
   [3]
   [4]
   [5]]
   matrix_b:
   [[4,3,2,1]]
   */
//-- 计算两个矩阵的 GeMM 并获取相同的结果
   matrix_c.GeMM(matrix_a, matrix_b, 1, 0);
   Print("matrix_c:\n ", matrix_c);
```

© 2000-2025, MetaQuotes Ltd.

```
   /*
   matrix_c:
    [[4,3,2,1]
    [8,6,4,2]
    [12,9,6,3]
    [16,12,8,4]
    [20,15,10,5]]
   */
  }
```

参见

MatMul

© 2000-2025, MetaQuotes Ltd.

# Power

将方阵提升整数幂。

```
matrix matrix::Power(
  const int  power      // 幂
   );
```

参数

*power*

[输入] 指数可以是任何整数、正数、负数或零。

返回值

矩阵。

注意

生成的矩阵与原始矩阵的大小相同。当升幂为 0 时，返回单位矩阵。正幂 n 表示原始矩阵自身乘以 n 次。负幂 -n 表示原始矩阵先倒置，然后将倒置矩阵乘以自身 n 次。

以 MQL5 实现的矩阵升幂的简单算法：

```
bool MatrixPower(matrix& c, const matrix& a, const int power)
  {
//--- 矩阵必须是正阵
   if(a.Rows()!=a.Cols())
      return(false);
//--- 结果矩阵的大小完全相同
   ulong  rows=a.Rows();
   ulong  cols=a.Cols();
   matrix result(rows,cols);
//--- 当幂为零时，返回单位矩阵
   if(power==0)
      result.Identity();
   else
     {
      //--- 对于负值，首先反转矩阵
      if(power<0)
        {
         matrix inverted=a.Inv();
         result=inverted;
         for(int i=-1; i>power; i--)
            result=result.MatMul(inverted);
        }
      else
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         result=a;
         for(int i=1; i<power; i++)
            result=result.MatMul(a);
        }
     }
//---
   c=result;
   return(true);
  }
```

MQL5 示例：

```
  matrix i= {{0, 1}, {-1, 0}};
  Print("i:\n", i);

  Print("i.Power(3):\n", i.Power(3));

  Print("i.Power(0):\n", i.Power(0));

  Print("i.Power(-3):\n", i.Power(-3));

  /*
  i:
  [[0,1]
   [-1,0]]

  i.Power(3):
  [[0,-1]
   [1,0]]

  i.Power(0):
  [[1,0]
   [0,1]]

  i.Power(-3):
  [[0, -1]
   [1,0]]
  */
```

Python 示例：

```
import numpy as np
from numpy.linalg import matrix_power

# matrix equiv. of the imaginary unit
i = np.array([[0, 1], [-1, 0]])
print("i:\n",i)
```

© 2000-2025, MetaQuotes Ltd.

```
# should = -i
print("matrix_power(i, 3) :\n",matrix_power(i, 3) )

print("matrix_power(i, 0):\n",matrix_power(i, 0))

# should = 1/(-i) = i, but w/ f.p. elements
print("matrix_power(i, -3):\n",matrix_power(i, -3))

i:
 [[ 0  1]
 [-1  0]]

matrix_power(i, 3) :
 [[ 0 -1]
 [ 1  0]]

matrix_power(i, 0):
 [[1 0]
 [0 1]]

matrix_power(i, -3):
 [[ 0.  1.]
 [-1.  0.]]
```

© 2000-2025, MetaQuotes Ltd.

# Dot

两个向量的点积。

```
double vector::Dot(
  const vector&  b      // 第二个向量
   );
```

参数

*b*

[输入] 向量。

返回值

标量。

注意

两个矩阵的点积是矩阵乘积 matrix::MatMul()。

以 MQL5 实现的向量标量乘积的简单算法：

```
double VectorDot(const vector& vector_a, const vector& vector_b)
  {
   double dot=0;

   if(vector_a.Size()==vector_b.Size())
     {
      for(ulong i=0; i<vector_a.Size(); i++)
         dot+=vector_a[i]*vector_b[i];
     }

   return(dot);
  }
```

MQL5 示例：

```
   for(ulong i=0; i<rows; i++)
     {
      vector v1=a.Row(i);
      for(ulong j=0; j<cols; j++)
        {
         vector v2=b.Row(j);
         result[i][j]=v1.Dot(v2);
        }
     }
```

© 2000-2025, MetaQuotes Ltd.

Python 示例：

```
import numpy as np

a = [1, 0, 0, 1]
b = [4, 1, 2, 2]
print(np.dot(a, b))

>>> 6
```

© 2000-2025, MetaQuotes Ltd.

# Kron

返回两个矩阵的克罗内克 (Kronecker)乘积，矩阵和向量，向量和矩阵、或两个向量。

```
matrix matrix::Kron(
  const matrix&  b      // 第二个矩阵
   );

matrix matrix::Kron(
  const vector&  b      // 向量
   );

matrix vector::Kron(
  const matrix&  b      // 矩阵
   );

matrix vector::Kron(
  const vector&  b      // 第二个向量
   );
```

参数

*b*

[输入] 第二个矩阵。

返回值

矩阵。

注意

克罗内克 (Kronecker)乘积也称为块矩阵乘法。

以 MQL5 实现的两个矩阵克罗内克 (Kronecker)乘积的简单算法：

```
matrix MatrixKronecker(const matrix& matrix_a,const matrix& matrix_b)
  {
   ulong  M=matrix_a.Rows();
   ulong  N=matrix_a.Cols();
   ulong  P=matrix_b.Rows();
   ulong  Q=matrix_b.Cols();
   matrix matrix_c(M*P,N*Q);

   for(ulong m=0; m<M; m++)
      for(ulong n=0; n<N; n++)
         for(ulong p=0; p<P; p++)
            for(ulong q=0; q<Q; q++)
```

© 2000-2025, MetaQuotes Ltd.

```
                matrix_c[m*P+p][n*Q+q]=matrix_a[m][n] * matrix_b[p][q];

    return(matrix_c);
   }
```

MQL5 示例：

```
   matrix a={{1,2,3},{4,5,6}};
   matrix b=matrix::Identity(2,2);
   vector v={1,2};

   Print(a.Kron(b));
   Print(a.Kron(v));

  /*
   [[1,0,2,0,3,0]
    [0,1,0,2,0,3]
    [4,0,5,0,6,0]
    [0,4,0,5,0,6]]

   [[1,2,2,4,3,6]
    [4,8,5,10,6,12]]
  */
```

Python 示例：

```
import numpy as np

A = np.arange(1,7).reshape(2,3)
B = np.identity(2)
V = [1,2]
print(np.kron(A, B))
print("")
print(np.kron(A, V))

[[1. 0. 2. 0. 3. 0.]
 [0. 1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6. 0.]
 [0. 4. 0. 5. 0. 6.]]

[[ 1  2  2  4  3  6]
 [ 4  8  5 10  6 12]]
```

© 2000-2025, MetaQuotes Ltd.

# Inner

两个矩阵的内积。

```
matrix matrix::Inner(
  const matrix&  b      // 第二个矩阵
   );
```

参数

*b*

[输入] 矩阵。

返回值

矩阵。

注意

两个向量的内积是两个向量的点积 vector::Dot()。

以 MQL5 实现的两个矩阵内积的简单算法：

```
bool MatrixInner(matrix& c, const matrix& a, const matrix& b)
  {
//--- 列数必须相等
   if(a.Cols()!=b.Cols())
      return(false);
//--- 所得矩阵的大小取决于每个矩阵中向量的数量
   ulong  rows=a.Rows();
   ulong  cols=b.Rows();
   matrix result(rows,cols);
//---
   for(ulong i=0; i<rows; i++)
     {
      vector v1=a.Row(i);
      for(ulong j=0; j<cols; j++)
        {
         vector v2=b.Row(j);
         result[i][j]=v1.Dot(v2);
        }
     }
//---
   c=result;
   return(true);
  }
```

MQL5 示例：

© 2000-2025, MetaQuotes Ltd.

```
   matrix a={{0,1,2},{3,4,5}};
   matrix b={{0,1,2},{3,4,5},{6,7,8}};
   matrix c=a.Inner(b);
   Print(c);
   matrix a1={{0,1,2}};
   matrix c1=a1.Inner(b);
   Print(c1);

  /*
  [[5,14,23]
  [14,50,86]]
  [[5,14,23]]
  */
```

Python 示例：

```
import numpy as np

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
A1= np.arange(3)
print(np.inner(A, B))
print("");
print(np.inner(A1, B))

import numpy as np

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
A1= np.arange(3)
print(np.inner(A, B))
print("");
print(np.inner(A1, B))
```

© 2000-2025, MetaQuotes Ltd.

# Outer

计算两个矩阵或两个向量的外积。

```
matrix matrix::Outer(
  const matrix&  b      // 第二个矩阵
   );

matrix vector::Outer(
  const vector&  b      // 第二个向量
   );
```

参数

*b*

[输入] 矩阵。

返回值

矩阵。

注意

外积，如克罗内克（Kronecker）乘积，也是一个区块矩阵（和向量）乘法。

以 MQL5 实现的两个矩阵外积的简单算法：

```
matrix MatrixOuter(const matrix& matrix_a, const matrix& matrix_b)
  {
//--- 生成的矩阵大小取决于矩阵的大小
   ulong  rows=matrix_a.Rows()*matrix_a.Cols();
   ulong  cols=matrix_b.Rows()*matrix_b.Cols();
   matrix matrix_c(rows,cols);
   ulong  cols_a=matrix_a.Cols();
   ulong  cols_b=matrix_b.Cols();
//---
   for(ulong i=0; i<rows; i++)
     {
      ulong row_a=i/cols_a;
      ulong col_a=i%cols_a;
      for(ulong j=0; j<cols; j++)
        {
         ulong row_b=j/cols_b;
         ulong col_b=j%cols_b;
         matrix_c[i][j]=matrix_a[row_a][col_a] * matrix_b[row_b][col_b];
        }
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//---
   return(matrix_c);
  }
```

MQL5 示例：

```
   vector vector_a={0,1,2,3,4,5};
   vector vector_b={0,1,2,3,4,5,6};
   Print("vector_a.Outer\n",vector_a.Outer(vector_b));
   Print("vector_a.Kron\n",vector_a.Kron(vector_b));

   matrix matrix_a={{0,1,2},{3,4,5}};
   matrix matrix_b={{0,1,2},{3,4,5},{6,7,8}};
   Print("matrix_a.Outer\n",matrix_a.Outer(matrix_b));
   Print("matrix_a.Kron\n",matrix_a.Kron(matrix_b));

  /*
   vector_a.Outer
   [[0,0,0,0,0,0,0]
    [0,1,2,3,4,5,6]
    [0,2,4,6,8,10,12]
    [0,3,6,9,12,15,18]
    [0,4,8,12,16,20,24]
    [0,5,10,15,20,25,30]]
   vector_a.Kron
   [[0,0,0,0,0,0,0,0,1,2,3,4,5,6,0,2,4,6,8,10,12,0,3,6,9,12,15,18,0,4,8,12,16,20,24,0,
   matrix_a.Outer
   [[0,0,0,0,0,0,0,0,0]
    [0,1,2,3,4,5,6,7,8]
    [0,2,4,6,8,10,12,14,16]
    [0,3,6,9,12,15,18,21,24]
    [0,4,8,12,16,20,24,28,32]
    [0,5,10,15,20,25,30,35,40]]
   matrix_a.Kron
   [[0,0,0,0,1,2,0,2,4]
    [0,0,0,3,4,5,6,8,10]
    [0,0,0,6,7,8,12,14,16]
    [0,3,6,0,4,8,0,5,10]
    [9,12,15,12,16,20,15,20,25]
    [18,21,24,24,28,32,30,35,40]]
   */
```

Python 示例：

```
import numpy as np

A = np.arange(6)
```

© 2000-2025, MetaQuotes Ltd.

```
B = np.arange(7)
print("np.outer")
print(np.outer(A, B))
print("np.kron")
print(np.kron(A, B))

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
print("np.outer")
print(np.outer(A, B))
print("np.kron")

np.outer
[[ 0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6]
 [ 0  2  4  6  8 10 12]
 [ 0  3  6  9 12 15 18]
 [ 0  4  8 12 16 20 24]
 [ 0  5 10 15 20 25 30]]
np.kron
[ 0  0  0  0  0  0  0  0  1  2  3  4  5  6  0  2  4  6  8 10 12  0  3  6
  9 12 15 18  0  4  8 12 16 20 24  0  5 10 15 20 25 30]
np.outer
[[ 0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8]
 [ 0  2  4  6  8 10 12 14 16]
 [ 0  3  6  9 12 15 18 21 24]
 [ 0  4  8 12 16 20 24 28 32]
 [ 0  5 10 15 20 25 30 35 40]]
np.kron
[[ 0  0  0  0  1  2  0  2  4]
 [ 0  0  0  3  4  5  6  8 10]
 [ 0  0  0  6  7  8 12 14 16]
 [ 0  3  6  0  4  8  0  5 10]
 [ 9 12 15 12 16 20 15 20 25]
 [18 21 24 24 28 32 30 35 40]]
```

© 2000-2025, MetaQuotes Ltd.

# CorrCoef

计算皮尔逊（Pearson）相关系数（线性相关系数）。

```
matrix matrix::CorrCoef(
  const bool     rowvar=true  // 所观测向量的行或列
);

scalar vector::CorrCoef(
  const vector&  b            // 第二个向量
);
```

返回值

皮尔逊（Pearson）乘积矩相关系数。

注意

相关系数在 [-1, 1] 范围内。

由于浮点舍入，生成的数组可能不是埃尔米特（Hermitian）数组，对角元素也许不是 1，并且元素也许不满足不等式 abs(a) <= 1。实部和虚部被剪裁到区间 [-1, 1]，以便试图改善这种情况，但在复杂情况下并无多大帮助。

以 MQL5 实现的计算两个向量相关系数的简单算法：

```
double VectorCorrelation(const vector& vector_x,const vector& vector_y)
  {
   ulong n=vector_x.Size()<vector_y.Size() ? vector_x.Size() : vector_y.Size();
   if(n<=1)
      return(0);

   ulong  i;
   double xmean=0;
   double ymean=0;
   for(i=0; i<n; i++)
     {
      if(!MathIsValidNumber(vector_x[i]))
         return(0);
      if(!MathIsValidNumber(vector_y[i]))
         return(0);
      xmean+=vector_x[i];
      ymean+=vector_y[i];
     }
   xmean/=(double)n;
   ymean/=(double)n;
```

© 2000-2025, MetaQuotes Ltd.

```
   double s=0;
   double xv=0;
   double yv=0;
   double t1=0;
   double t2=0;
//--- 计算
   s=0;
   for(i=0; i<n; i++)
     {
      t1=vector_x[i]-xmean;
      t2=vector_y[i]-ymean;
      xv+=t1*t1;
      yv+=t2*t2;
      s+=t1*t2;
     }
//--- 检查
   if(xv==0 || yv==0)
      return(0);
//--- 返回结果
   return(s/(MathSqrt(xv)*MathSqrt(yv)));
  }
```

MQL5 示例：

```
   vectorf vector_a={1,2,3,4,5};
   vectorf vector_b={0,1,0.5,2,2.5};
   Print("vectors correlation ",vector_a.CorrCoef(vector_b));
//---
   matrixf matrix_a={{1,2,3,4,5},
                     {0,1,0.5,2,2.5}};
   Print("matrix rows correlation\n",matrix_a.CorrCoef());
   matrixf matrix_a2=matrix_a.Transpose();
   Print("transposed matrix cols correlation\n",matrix_a2.CorrCoef(false));
   matrixf matrix_a3={{1.0f, 2.0f, 3.0f, 4.0f, 5.0f},
                      {0.0f, 1.0f, 0.5f, 2.0f, 2.5f},
                      {0.1f, 1.0f, 2.0f, 1.0f, 0.3f}};
   Print("rows correlation\n",matrix_a3.CorrCoef());
   Print("cols correlation\n",matrix_a3.CorrCoef(false));

  /*
   vectors correlation 0.9149913787841797
   matrix rows correlation
   [[1,0.91499138]
    [0.91499138,1]]
   transposed matrix cols correlation
   [[1,0.91499138]
    [0.91499138,1]]
   rows correlation
```

© 2000-2025, MetaQuotes Ltd.

```
   [[1,0.91499138,0.08474271]
    [0.91499138,1,-0.17123166]
    [0.08474271,-0.17123166,1]]
   cols correlation
   [[1,0.99587059,0.85375023,0.91129309,0.83773589]
    [0.99587059,1,0.80295509,0.94491106,0.88385159]
    [0.85375023,0.80295509,1,0.56362146,0.43088508]
    [0.91129309,0.94491106,0.56362146,1,0.98827404]
    [0.83773589,0.88385159,0.43088508,0.98827404,1]]
   */
```

Python 示例：

```
import numpy as np
va=[1,2,3,4,5]
vb=[0,1,0.5,2,2.5]
print("vectors correlation")
print(np.corrcoef(va,vb))

ma=np.zeros((2,5))
ma[0,:]=va
ma[1,:]=vb
print("matrix rows correlation")
print(np.corrcoef(ma))
print("transposed matrix cols correlation")
print(np.corrcoef(np.transpose(ma),rowvar=False))
print("")

ma1=[[1,2,3,4,5],[0,1,0.5,2,2.5],[0.1,1,0.2,1,0.3]]
print("rows correlation\n",np.corrcoef(ma1))
print("cols correlation\n",np.corrcoef(ma1,rowvar=False))

transposed matrix cols correlation
[[1.         0.91499142]
 [0.91499142 1.        ]]

rows correlation
 [[1.         0.91499142 0.1424941 ]
 [0.91499142 1.         0.39657517]
 [0.1424941  0.39657517 1.        ]]
cols correlation
 [[1.         0.99587059 0.98226063 0.91129318 0.83773586]
 [0.99587059 1.         0.99522839 0.94491118 0.88385151]
 [0.98226063 0.99522839 1.         0.97234063 0.92527551]
 [0.91129318 0.94491118 0.97234063 1.         0.98827406]
 [0.83773586 0.88385151 0.92527551 0.98827406 1.        ]]
```

© 2000-2025, MetaQuotes Ltd.

## Cov

计算协方差矩阵。

```
matrix matrix::Cov(
  const bool     rowvar=true  // 观测值的行或列向量
);

matrix matrix::Cov(
  const bool     rowvar,      // 观测值的行或列向量
  const int      ddof         // delta自由度
);

matrix vector::Cov(
  const vector& b             // 第二个向量
);

matrix vector::Cov(
  const vector& b,            // 第二个向量
  const int      ddof         // delta自由度
);
```

参数

*rowwar*

[in] 如果 rowvar 为 true（默认值），则每一行表示一个变量，列中有观测值。否则，关系将被转置：每一列代表一个变量，而行包含观测值。

*b*

[in] 第二个观测向量。

*ddof*

[in] "delta 自由度"：计算中使用的除数是 N-ddof，其中 N 表示元素的数量。默认情况下，ddof 为 1。

请注意

计算协方差矩阵。

使用 MQL5 计算两个向量协方差矩阵的简单算法：

```
bool VectorCovariation(const vector& vector_a,const vector& vector_b,matrix& matrix_c)
  {
   int i,j;
   int m=2;
   int n=(int)(vector_a.Size()<vector_b.Size()?vector_a.Size():vector_b.Size());
//--- 检查
   if(n<=1)
      return(false);
   for(i=0; i<n; i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       if(!MathIsValidNumber(vector_a[i]))
          return(false);
       if(!MathIsValidNumber(vector_b[i]))
          return(false);
      }
//---
   matrix matrix_x(2,n);
   matrix_x.Row(vector_a,0);
   matrix_x.Row(vector_b,1);
   vector t=vector::Zeros(m);
//--- 计算
   for(i=0; i<m; i++)
      for(j=0; j<n; j++)
         t[i]+=matrix_x[i][j]/double(n);
   for(i=0; i<m; i++)
      for(j=0; j<n; j++)
         matrix_x[i][j]-=t[i];
//--- syrk C=alpha*A^H*A+beta*C (beta=0 并且不作考虑)
   matrix_c=matrix::Zeros(m,m);
   for(i=0; i<m; i++)
     {
      for(j=0; j<n; j++)
        {
         double v=matrix_x[i][j]/(n-1);
         for(int i_=i; i_<m; i_++)
            matrix_c[i][i_]+=v*matrix_x[i_][j];
        }
     }
//--- 强制对称性
   for(i=0; i<m-1; i++)
      for(j=i+1; j<m; j++)
         matrix_c[j][i]=matrix_c[i][j];
//---
   return(true);
  }
```

MQL5 示例：

```
   matrix matrix_a={{3,-2.1},{1.1,-1},{0.12,4.3}};
   Print("covariation cols\n",matrix_a.Cov(false));
   Print("covariation rows\n",matrix_a.Cov());

   vector vector_a=matrix_a.Col(0);
   vector vector_b=matrix_a.Col(1);
   Print("covariation vectors\n",vector_a.Cov(vector_b));

  /*
```

© 2000-2025, MetaQuotes Ltd.

```
   covariation cols
   [[2.144133333333333,-4.286]
    [-4.286,11.71]]
   covariation rows
   [[13.005,5.355,-10.659]
    [5.355,2.205,-4.389]
    [-10.659,-4.389,8.736199999999998]]
   covariation vectors
   [[2.144133333333333,-4.286]
    [-4.286,11.71]]
  */
```

Python 示例：

```
import numpy as np
matrix_a=np.array([[3,-2.1],[1.1,-1],[0.12,4.3]])
matrix_c=np.cov(matrix_a,rowvar=False)
print("covariation cols\n",matrix_c)
matrix_c2=np.cov(matrix_a)
print("covariation rows\n",matrix_c2)

vector_a=matrix_a[:,0]
vector_b=matrix_a[:,1]
matrix_c3=np.cov(vector_a,vector_b)
print("covariation vectors\n",matrix_c3)

covariation cols
 [[ 2.14413333 -4.286     ]
 [-4.286      11.71      ]]
covariation rows
 [[ 13.005    5.355  -10.659 ]
 [  5.355    2.205   -4.389 ]
 [-10.659   -4.389    8.7362]]
covariation vectors
 [[ 2.14413333 -4.286     ]
 [-4.286      11.71      ]]
```

© 2000-2025, MetaQuotes Ltd.

# Correlate

计算两个向量的互相关。

```
vector vector::Correlate(
  const vector&           v,        // 向量
  ENUM_VECTOR_CONVOLVE    mode      // 模式
   );
```

参数

*v*

[输入] 第二个向量。

*mode*

[输入] “mode”参数判定线性卷积计算模式。数值来自 ENUM_VECTOR_CONVOLVE 枚举。

返回值

两个向量的互相关。

注意

“mode”参数判定线性卷积计算模式。

以 MQL5 实现的计算两个向量相关系数的简单算法：

```
vector VectorCrossCorrelationFull(const vector& a,const vector& b)
  {
   int     m=(int)a.Size();
   int     n=(int)b.Size();
   int     size=m+n-1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
      for(int i_=i; i_<i+m; i_++)
         c[i_]+=b[n-i-1]*a[i_-i];

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorCrossCorrelationSame(const vector& a,const vector& b)
  {
   int     m=(int)a.Size();
   int     n=(int)b.Size();
   int     size=MathMax(m,n);
   vector c=vector::Zeros(size);
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-size/2+1;
         if(k>=0 && k<size)
            c[k]+=b[n-i-1]*a[i_-i];
        }
     }

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorCrossCorrelationValid(const vector& a,const vector& b)
  {
   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=MathMax(m,n)-MathMin(m,n)+1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-n+1;
         if(k>=0 && k<size)
            c[k]+=b[n-i-1]*a[i_-i];
        }
     }

   return(c);
  }
```

MQL5 示例：

```
   vector a={1,2,3,4,5};
   vector b={0,1,0.5};

   Print("full\n",a.Correlate(b,VECTOR_CONVOLVE_FULL));
   Print("same\n",a.Correlate(b,VECTOR_CONVOLVE_SAME));
   Print("valid\n",a.Correlate(b,VECTOR_CONVOLVE_VALID));
   Print("full\n",b.Correlate(a,VECTOR_CONVOLVE_FULL));

  /*
   full
   [0.5,2,3.5,5,6.5,5,0]
```

© 2000-2025, MetaQuotes Ltd.

```
   same
   [2,3.5,5,6.5,5]
   valid
   [3.5,5,6.5]
   full
   [0,5,6.5,5,3.5,2,0.5]
  */
```

Python 示例：

```
import numpy as np
a=[1,2,3,4,5]
b=[0,1,0.5]

print("full\n",np.correlate(a,b,'full'))
print("same\n",np.correlate(a,b,'same'));
print("valid\n",np.correlate(a,b,'valid'));
print("full\n",np.correlate(b,a,'full'))

full
 [0.5 2.  3.5 5.  6.5 5.  0. ]
same
 [2.  3.5 5.  6.5 5. ]
valid
 [3.5 5.  6.5]
full
 [0.  5.  6.5 5.  3.5 2.  0.5]
```

© 2000-2025, MetaQuotes Ltd.

# Convolve

返回两个向量的离散线性卷积

```
vector vector::Convolve(
  const vector&            v,          // 向量
  ENUM_VECTOR_CONVOLVE     mode        // 模式
   );
```

参数

*v*

[输出] 第二个向量。

*mode*

[输入] “mode”参数判定线性卷积计算模式 ENUM_VECTOR_CONVOLVE。

返回值

两个向量的离散，线性卷积。

以 MQL5 实现的计算两个向量卷积的简单算法：

```
vector VectorConvolutionFull(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionFull(b,a));

   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=m+n-1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
      for(int i_=i; i_<i+m; i_++)
         c[i_]+=b[i]*a[i_-i];

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorConvolutionSame(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionSame(b,a));

   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=MathMax(m,n);
```

© 2000-2025, MetaQuotes Ltd.

```
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-size/2+1;
         if(k>=0 && k<size)
            c[k]+=b[i]*a[i_-i];
        }
     }

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorConvolutionValid(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionValid(b,a));

   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=MathMax(m,n)-MathMin(m,n)+1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-n+1;
         if(k>=0 && k<size)
            c[k]+=b[i]*a[i_-i];
        }
     }

   return(c);
  }
```

MQL5 示例：

```
  vector a= {1, 2, 3, 4, 5};
  vector b= {0, 1, 0.5};

  Print("full\n", a.Convolve(b, VECTOR_CONVOLVE_FULL));
  Print("same\n", a.Convolve(b, VECTOR_CONVOLVE_SAME));
  Print("valid\n", a.Convolve(b, VECTOR_CONVOLVE_VALID));
```

© 2000-2025, MetaQuotes Ltd.

```
/*
 full
 [0,1,2.5,4,5.5,7,2.5]
 same
 [1,2.5,4,5.5,7]
 valid
 [2.5,4,5.5]
*/
```

Python 示例：

```
import numpy as np
a=[1,2,3,4,5]
b=[0,1,0.5]

print("full\n",np.convolve(a,b,'full'))
print("same\n",np.convolve(a,b,'same'));
print("valid\n",np.convolve(a,b,'valid'));


full
 [0.  1.  2.5 4.  5.5 7.  2.5]
same
 [1.  2.5 4.  5.5 7. ]
valid
 [2.5 4.  5.5]
```

© 2000-2025, MetaQuotes Ltd.

# 矩阵变换

矩阵分解可用于以下情况：

- 作为求解线性方程组的中间步骤
- 用于矩阵反演
- 计算行列式时
- 查找矩阵的本征值和本征向量时
- 计算矩阵的解析函数时
- 使用最小二乘法时
- 在微分方程的数值解中

根据问题采用不同的矩阵分解类型。

| 函数 | 动作 |
| --- | --- |
| Cholesky | 计算乔列斯基分解 |
| Eig | 计算方阵的本征值和右本征向量 |
| EigVals | 计算一般矩阵的本征值 |
| LU | 矩阵的 LU 因数分解为下三角矩阵和上三角矩阵的乘积 |
| LUP | 具有部分枢轴的 LUP 因数分解，仅指具有行排列的 LU 分解：PA=LU |
| QR | 计算矩阵的 QR 分解 |
| SVD | 奇异值分解 |

© 2000-2025, MetaQuotes Ltd.

# Cholesky

计算乔列斯基 (Cholesky)分解。

```
bool matrix::Cholesky(
  matrix&  L      // 矩阵
   );
```

参数

*L* 键

[输出] 下三角矩阵。

返回值

成功时返回 true，否则返回 false。

注意

返回方阵 a 的乔列斯基分解，L*L.H，其中 L 是下三角形和，且 H 是共轭转置运算符 (如果 a 是实值，则为普通转置)。a 必须是埃尔米特 (Hermitian，如果实值是对称的)，且正定。不执行任何检查来验证 a 是否是埃尔米特 (Hermitian)。此外，仅用到 a 的下三角形和对角线元素。仅有 L 实际返回。

举例

```
matrix matrix_a= {{5.7998084, -2.1825367}, {-2.1825367, 9.85910595}};
matrix matrix_l;
Print("matrix_a\n", matrix_a);

matrix_a.Cholesky(matrix_l);
Print("matrix_l\n", matrix_l);
Print("check\n", matrix_l.MatMul(matrix_l.Transpose()));

/*
matrix_a
[[5.7998084,-2.1825367]
 [-2.1825367,9.85910595]]
matrix_l
[[2.408279136645086,0]
 [-0.9062640068544704,3.006291985133859]]
check
[[5.7998084,-2.1825367]
 [-2.1825367,9.85910595]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Eig

计算方阵的本征值和右本征向量。

```
bool matrix::Eig(
  matrix&  eigen_vectors,          // 本征向量矩阵
  vector&  eigen_values            // 本征向量向量
   );
```

特征值和特征向量的复解

```
bool matrix::Eig(
  matrix<complex>& eigen_vectors, // 特征向量矩阵
  vector<complex>& eigen_values   // 特征值向量
   );
```

参数

*eigen_vectors*

[输出] 垂直本征向量矩阵。

*eigen_values*

[输出] 本征值向量。

返回值

成功时返回 true，否则返回 false。

注意

如果计算特征值时遇到复解，则停止计算，并将错误代码设置为4019 (ERR_MATH_OVERFLOW)。使用Eig方法的复数重载在复数空间中获得完整的解决方案

如果一个复特征值的虚部等于零，那么它就是一个实特征值。这可以在下面的示例中看到。

示例：

```
void OnStart()
  {
   matrix matrix_a =
     {
        {-3.474589, 1.106384, -9.091977,-3.925227 },
        {-5.522139, 2.366887,-15.162351,-6.357512 },
        { 8.394926,-2.960067, 22.292115, 9.524129 },
        { 7.803242,-2.080287, 19.217706, 8.186645 }
     };

   matrix eigen_vectors;
   vector eigen_values;

   bool res=matrix_a.Eig(eigen_vectors, eigen_values);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("res=",res,"  error=",GetLastError());
   Print("Eigen vectors:\n",eigen_vectors, "\nEigen Values:\n",eigen_values);

//--- 检查正确性 A * v = lambda * v
   int vectors=0;
   for(ulong n=0; n<eigen_values.Size(); n++)
     {
      vector eigen_vector=eigen_vectors.Col(n);
      vector vector_d1   =eigen_vector*eigen_values[n];
      vector vector_d2   =matrix_a.MatMul(eigen_vector);

      ulong errors=vector_d1.Compare(vector_d2,1e-13);
      if(errors==0)
         vectors++;
     }
   Print("vectors=",vectors);

//--- 复解
   matrix<complex> eigen_vectors_c;
   vector<complex> eigen_values_c;
   ResetLastError();
   res=matrix_a.Eig(eigen_vectors_c,eigen_values_c);
   Print("res=",res,"  error=",GetLastError());
   Print("Eigen vectors:\n",eigen_vectors_c, "\nEigen Values:\n",eigen_values_c);

//--- 检查正确性 A * v = lambda * v
   matrixc matrix_c;
   matrix_c.Assign(matrix_a);
   vectors=0;
   for(ulong n=0; n<eigen_values_c.Size(); n++)
     {
      vectorc eigen_vector_c=eigen_vectors_c.Col(n);
      vectorc vector_c1     =eigen_vector_c*eigen_values_c[n];
      vectorc vector_c2     =matrix_c.MatMul(eigen_vector_c);

      ulong errors=vector_c1.Compare(vector_c2,1e-13);
      if(errors==0)
         vectors++;
     }
   Print("vectors=",vectors);
  }
/* Result
   res=true  error=4019
   Eigen vectors:
   [[0.2649667608713664]
    [0.4488818803991876]
    [-0.6527335897527492]
    [-0.5497604331807768]]
   Eigen Values:
```

© 2000-2025, MetaQuotes Ltd.

```
   [28.94158645962942]
   vectors=1
   res=true  error=0
   Eigen vectors:
   [[(0.2649667608713664,0),(0.2227392219204762,0.3745470492013296),(0.403285439089771
    [(0.4488818803991876,0),(-0.2613452530342438,-0.3685707727819327),(-0.396850612637
    [(-0.6527335897527492,0),(-0.3418479634708521,-0.3299830378162041),(-0.355302103111
    [(-0.5497604331807768,0),(0.523227993452828,0.3262508584080381),(0.351283559614366
   Eigen Values:
   [(28.94158645962942,0),(0.01022501104810897,0.02954980822331488),(0.010225011048108
   vectors=3
*/
```

© 2000-2025, MetaQuotes Ltd.

# EigVals

计算一般矩阵的本征值。

```
bool matrix::EigVals(
  vector&  eigen_values      // 本征向量
   );
```

参数

*eigen_values*

[输出] 右本征值的向量。

返回值

成功时返回 true，否则返回 false。

注意

EigVals 和 Eig 之间的唯一区别是 EigVals 不计算本征向量，它只计算本征值。

© 2000-2025, MetaQuotes Ltd.

# LU

矩阵的 LU 因数分解为下三角矩阵和上三角矩阵的乘积。

```
bool matrix::LU(
  matrix&  L,      // 下三角矩阵
  matrix&  U       // 上三角矩阵
   );
```

参数

*L* 键

[输出] 下三角矩阵。

*U*

[输出] 上三角矩阵。

返回值

成功时返回 true，否则返回 false。

举例

```
   matrix matrix_a={{1,2,3,4},
                    {5,2,6,7},
                    {8,9,3,10},
                    {11,12,14,4}};
   matrix matrix_l,matrix_u;
//--- LU 分解
   matrix_a.LU(matrix_l,matrix_u);
   Print("matrix_l\n",matrix_l);
   Print("matrix_u\n",matrix_u);
//--- 验算是否 A = L * U
   Print("check\n",matrix_l.MatMul(matrix_u));


   /*
   matrix_l
   [[1,0,0,0]
    [5,1,0,0]
    [8,0.875,1,0]
    [11,1.25,0.5904761904761905,1]]
   matrix_u
   [[1,2,3,4]
    [0,-8,-9,-13]
    [0,0,-13.125,-10.625]
    [0,0,0,-17.47619047619047]]
   check
   [[1,2,3,4]
    [5,2,6,7]
```

© 2000-2025, MetaQuotes Ltd.

```
    [8,9,3,10]
    [11,12,14,4]]
   */
```

© 2000-2025, MetaQuotes Ltd.

# LUP

具有部分轴的 LUP 因数分解，仅指具有行排列的 LU 分解：PA=LU。

```
bool  LUP(
  matrix&  L,      // 下三角矩阵
  matrix&  U,      // 上三角矩阵
  matrix&  P       // 排列矩阵
   );
```

参数

*L 键*

[输出] 下三角矩阵。

*U*

[输出] 上三角矩阵。

*P*

[out] Permutations matrix

返回值

成功时返回 true，否则返回 false。

举例

```
   matrix matrix_a={{1,2,3,4},
                    {5,2,6,7},
                    {8,9,3,10},
                    {11,12,14,4}};
   matrix matrix_l,matrix_u,matrix_p;
//--- LUP 分解
   matrix_a.LUP(matrix_l,matrix_u,matrix_p);
   Print("matrix_l\n",matrix_l);
   Print("matrix_u\n",matrix_u);
   Print("matrix_p\n",matrix_p);
//--- 验算是否 P * A = L * U
   Print("P * A\n",matrix_p.MatMul(matrix_a));
   Print("L * U\n",matrix_l.MatMul(matrix_u));

   /*
   matrix_l
   [[1,0,0,0]
    [0.4545454545454545,1,0,0]
    [0.7272727272727273,-0.07894736842105282,1,0]
    [0.09090909090909091,-0.2631578947368421,-0.2262773722627738,1]]
   matrix_u
   [[11,12,14,4]
    [0,-3.454545454545454,-0.3636363636363633,5.181818181818182]
```

© 2000-2025, MetaQuotes Ltd.

```
   [0,0,-7.210526315789473,7.500000000000001]
   [0,0,0,6.697080291970803]]
  matrix_p
  [[0,0,0,1]
   [0,1,0,0]
   [0,0,1,0]
   [1,0,0,0]]
  P * A
  [[11,12,14,4]
   [5,2,6,7]
   [8,9,3,10]
   [1,2,3,4]]
  L * U
  [[11,12,14,4]
   [5,2,6,7]
   [8,9,3.000000000000001,10]
   [1,2,3,4]]
  */
```

© 2000-2025, MetaQuotes Ltd.

# QR

计算矩阵的 QR 分解。

```
bool  QR(
  matrix&  Q,      // 具有正交列的矩阵
  matrix&  R       // 上三角矩阵
   );
```

参数

*Q*

[输出] 具有正交列的矩阵。当模式= "完成"时，结果是一个正交/酉矩阵，这取决于 a 是否是实数/复数。在这种情况下，行列式可以是 +/- 1。如果输入数组中的维数大于 2，则返回拥有上述属性的矩阵堆栈。

*R 键*

[输出] 上三角矩阵。

返回值

成功时返回 true，否则返回 false。

举例

```
//---  A*x = b
  matrix A = {{0, 1}, {1, 1}, {1, 1}, {2, 1}};
  Print("A \n", A);
  vector b = {1, 2, 2, 3};
  Print("b \n", b);
//--- A = Q*R
  matrix q, r;
  A.QR(q, r);
  Print("q \n", q);
  Print("r \n", r);
  matrix qr=q.MatMul(r);
  Print("qr \n", qr);
  /*
  A
  [[0,1]
  [1,1]
  [1,1]
  [2,1]]
  b
  [1,2,2,3]
  q
  [[0.4082482904638631,-0.8164965809277259,-1.110223024625157e-16,-0.4082482904638631]
  [0.4625425214347352,-0.03745747856526496,0.7041241452319315,0.5374574785652647]
  [-0.5374574785652648,-0.03745747856526496,0.7041241452319316,-0.4625425214347352]
  [-0.5749149571305296,-0.5749149571305299,-0.09175170953613698,0.5749149571305296]]
```

© 2000-2025, MetaQuotes Ltd.

```
r
[[-1.224744871391589,-0.2415816237971962]
[-1.22474487139159,-1.466326495188786]
[1.224744871391589,1.316496580927726]
[1.224744871391589,0.2415816237971961]]
qr
[[-1.110223024625157e-16,1]
[1,0.9999999999999999]
[1,1]
[2,1]]
*/
```

© 2000-2025, MetaQuotes Ltd.

## SVD

奇异值分解。

```
bool matrix::SVD(
  matrix&  U,                    // 酉矩阵
  matrix&  V,                    // 酉矩阵
  vector&  singular_values       // 奇异值向量
   );
```

参数

*U*

[输出]m 阶酉矩阵，由左奇异向量组成。

*V*

[输出]n 阶酉矩阵，由右奇异向量组成。

*singular_values*

[输出]奇异值

返回值

成功时返回 true，否则返回 false。

举例

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a=a-4;
  Print("matrix a \n", a);
  a.Reshape(3, 3);
  matrix b=a;
  Print("matrix b \n", b);
//--- 执行 SVD 分解
  matrix U, V;
  vector singular_values;
  b.SVD(U, V, singular_values);
  Print("U \n", U);
  Print("V \n", V);
  Print("singular_values = ", singular_values);

// 检查模块
//--- U * 奇异值对角线 * V = A
  matrix matrix_s;
  matrix_s.Diag(singular_values);
  Print("matrix_s \n", matrix_s);
  matrix matrix_vt=V.Transpose();
  Print("matrix_vt \n", matrix_vt);
  matrix matrix_usvt=(U.MatMul(matrix_s)).MatMul(matrix_vt);
  Print("matrix_usvt \n", matrix_usvt);
```

© 2000-2025, MetaQuotes Ltd.

```
  ulong errors=(int)b.Compare(matrix_usvt, 1e-9);
  double res=(errors==0);
  Print("errors=", errors);

//---- 其它检查
  matrix U_Ut=U.MatMul(U.Transpose());
  Print("U_Ut \n", U_Ut);
  Print("Ut_U \n", (U.Transpose()).MatMul(U));

  matrix vt_V=matrix_vt.MatMul(V);
  Print("vt_V \n", vt_V);
  Print("V_vt \n", V.MatMul(matrix_vt));

  /*
  matrix a
  [[-4,-3,-2,-1,0,1,2,3,4]]
  matrix b
  [[-4,-3,-2]
   [-1,0,1]
   [2,3,4]]
  U
  [[-0.7071067811865474,0.5773502691896254,0.408248290463863]
   [-6.827109697437648e-17,0.5773502691896253,-0.8164965809277256]
   [0.7071067811865472,0.5773502691896255,0.4082482904638627]]
  V
  [[0.5773502691896258,-0.7071067811865474,-0.408248290463863]
   [0.5773502691896258,1.779939029415334e-16,0.8164965809277258]
   [0.5773502691896256,0.7071067811865474,-0.408248290463863]]
  singular_values = [7.348469228349533,2.449489742783175,3.277709923350408e-17]

  matrix_s
  [[7.348469228349533,0,0]
   [0,2.449489742783175,0]
   [0,0,3.277709923350408e-17]]
  matrix_vt
  [[0.5773502691896258,0.5773502691896258,0.5773502691896256]
   [-0.7071067811865474,1.779939029415334e-16,0.7071067811865474]
   [-0.408248290463863,0.8164965809277258,-0.408248290463863]]
  matrix_usvt
  [[-3.999999999999997,-2.999999999999999,-2]
   [-0.9999999999999981,-5.977974170712231e-17,0.9999999999999974]
   [2,2.999999999999999,3.999999999999996]]
  errors=0

  U_Ut
  [[0.9999999999999993,-1.665334536937735e-16,-1.665334536937735e-16]
   [-1.665334536937735e-16,0.9999999999999987,-5.551115123125783e-17]
   [-1.665334536937735e-16,-5.551115123125783e-17,0.9999999999999999]]
```

© 2000-2025, MetaQuotes Ltd.

```
  Ut_U
  [[0.9999999999999993,-5.551115123125783e-17,-1.110223024625157e-16]
   [-5.551115123125783e-17,0.9999999999999987,2.498001805406602e-16]
   [-1.110223024625157e-16,2.498001805406602e-16,0.999999999999999]]
  vt_V
  [[1,-5.551115123125783e-17,0]
   [-5.551115123125783e-17,0.9999999999999996,1.110223024625157e-16]
   [0,1.110223024625157e-16,0.9999999999999996]]
  V_vt
  [[0.9999999999999999,1.110223024625157e-16,1.942890293094024e-16]
   [1.110223024625157e-16,0.9999999999999998,1.665334536937735e-16]
   [1.942890293094024e-16,1.665334536937735e-16,0.9999999999999996]
  */
 }
```

© 2000-2025, MetaQuotes Ltd.

# 统计方法

计算矩阵和向量的描述性统计的方法。

| 函数 | 动作 |
| --- | --- |
| ArgMax | 返回最大值的索引 |
| ArgMin | 返回最小值的索引 |
| Max | 返回矩阵/向量中的最大值 |
| Min | 返回矩阵/向量中的最小值 |
| Ptp | 返回矩阵/向量或给定矩阵轴的数值范围 |
| Sum | 返回矩阵/矢量元素的总和，这些元素也可以针对给定轴（轴）执行 |
| Prod | 返回矩阵/矢量元素的乘积，也可以针对给定轴执行 |
| CumSum | 返回矩阵/矢量元素的累积总和，包括沿给定轴的矩阵/矢量元素 |
| CumProd | 返回矩阵/向量元素的累积乘积，包括沿给定轴的累积乘积 |
| Percentile | 返回矩阵/向量元素，或沿指定轴的元素值的指定百分位数 |
| Quantile | 返回矩阵/向量元素，或沿指定轴的元素值的指定分位数 |
| Median | 计算矩阵/向量元素的中位数 |
| Mean | 计算元素值的算术平均值 |
| Average | 计算矩阵/向量值的加权平均值 |
| Std | 返回矩阵/向量元素，或沿给定轴的元素值的标准偏差 |
| Var | 计算矩阵/向量元素值的方差 |
| LinearRegression | 计算包含计算的线性回归值的向量/矩阵 |

© 2000-2025, MetaQuotes Ltd.

# ArgMax

返回最大值的索引。

```
ulong vector::ArgMax();

ulong matrix::ArgMax();

vector matrix::ArgMax(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

最大值的索引。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_max=matrix_a.ArgMax(0);
   vector rows_max=matrix_a.ArgMax(1);
   ulong  matrix_max=matrix_a.ArgMax();

   Print("cols_max=",cols_max);
   Print("rows_max=",rows_max);
   Print("max index ",matrix_max,"  max value ",matrix_a.Flat(matrix_max));

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_max=[0,3,1]
   rows_max=[0,2,0,1]
   max index 5  max value 12.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# ArgMin

返回最小值的索引。

```
ulong vector::ArgMin();

ulong matrix::ArgMin();

vector matrix::ArgMin(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

最小值的索引。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_min=matrix_a.ArgMin(0);
   vector rows_min=matrix_a.ArgMin(1);
   ulong  matrix_min=matrix_a.ArgMin();

   Print("cols_min=",cols_min);
   Print("rows_min=",rows_min);
   Print("min index ",matrix_min,"  min value ",matrix_a.Flat(matrix_min));

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_min=[1,0,0]
   rows_min=[2,0,2,0]
   min index 3  min value 1.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# Max

返回矩阵/向量中的最大值。

```
double vector::Max();

double matrix::Max();

vector matrix::Max(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

矩阵/向量中的最大值。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_max=matrix_a.Max(0);
   vector rows_max=matrix_a.Max(1);
   double matrix_max=matrix_a.Max();

   Print("cols_max=",cols_max);
   Print("rows_max=",rows_max);
   Print("max value ",matrix_max);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_max=[10,11,12]
   rows_max=[10,12,6,11]
   max value 12.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# Min

返回矩阵/向量中的最小值。

```
double vector::Min();

double matrix::Min();

vector matrix::Min(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

矩阵/向量中的最小值。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_min=matrix_a.Min(0);
   vector rows_min=matrix_a.Min(1);
   double matrix_min=matrix_a.Min();

   Print("cols_min=",cols_min);
   Print("rows_min=",rows_min);
   Print("min value ",matrix_min);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_min=[1,3,2]
   rows_min=[2,1,4,7]
   min value 1.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# Ptp

返回矩阵/向量或给定矩阵轴的数值范围，等效于 Max()-Min()。Ptp-峰值到峰值。

```
double vector::Ptp();

double matrix::Ptp();

vector matrix::Ptp(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0—水平轴，1—垂直轴。

返回值

向量的数值范围 (最大值-最小值)

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_ptp=matrix_a.Ptp(0);
   vector rows_ptp=matrix_a.Ptp(1);
   double matrix_ptp=matrix_a.Ptp();

   Print("cols_ptp  ",cols_ptp);
   Print("rows_ptp  ",rows_ptp);
   Print("ptp value  ",matrix_ptp);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_ptp [9,8,10]
   rows_ptp [8,11,2,4]
   ptp value  11.0
   */
```

© 2000-2025, MetaQuotes Ltd.

## Sum

返回矩阵/矢量元素的总和，这些元素也可以针对给定轴（轴）执行。

```
double vector::Sum();

double matrix::Sum();

vector matrix::Sum(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

矩阵/矢量元素的总和，也可以针对给定轴（轴）执行。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_sum=matrix_a.Sum(0);
   vector rows_sum=matrix_a.Sum(1);
   double matrix_sum=matrix_a.Sum();

   Print("cols_sum=",cols_sum);
   Print("rows_sum=",rows_sum);
   Print("sum value ",matrix_sum);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_sum=[24,27,27]
   rows_sum=[15,21,15,27]
   sum value 78.0
   */
```

© 2000-2025, MetaQuotes Ltd.

# Prod

返回矩阵/矢量元素的乘积，该乘积也可以针对给定轴执行。

```
double vector::Prod(
  const double  initial=1      // initial multiplier
   );

double matrix::Prod(
  const double  initial=1      // initial multiplier
   );

vector matrix::Prod(
  const int     axis,          // axis
  const double  initial=1      // initial multiplier
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

*initial=1*

[输入] 初始乘数。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_prod=matrix_a.Prod(0);
   vector rows_prod=matrix_a.Prod(1);
   double matrix_prod=matrix_a.Prod();

   Print("cols_prod=",cols_prod);
   cols_prod=matrix_a.Prod(0,0.1);
   Print("cols_prod=",cols_prod);
   Print("rows_prod=",rows_prod);
   Print("prod value ",matrix_prod);

  /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_prod=[420,1320,864]
   cols_prod=[42,132,86.40000000000001]
   rows_prod=[60,96,120,693]
```

© 2000-2025, MetaQuotes Ltd.

```
   prod value 479001600.0
*/
```

© 2000-2025, MetaQuotes Ltd.

# CumSum

返回矩阵/矢量元素的累积总和，包括沿给定轴的元素。

```
vector vector::CumSum();

vector matrix::CumSum();

matrix matrix::CumSum(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

沿给定轴的元素的累积总和。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   matrix cols_cumsum=matrix_a.CumSum(0);
   matrix rows_cumsum=matrix_a.CumSum(1);
   vector cumsum_values=matrix_a.CumSum();

   Print("cols_cumsum\n",cols_cumsum);
   Print("rows_cumsum\n",rows_cumsum);
   Print("cumsum values  ",cumsum_values);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_cumsum
   [[10,3,2]
    [11,11,14]
    [17,16,18]
    [24,27,27]]
   rows_cumsum
   [[10,13,15]
    [1,9,21]
    [6,11,15]
```

© 2000-2025, MetaQuotes Ltd.

```
 [7,18,27]]
cumsum values  [10,13,15,16,24,36,42,47,51,58,69,78]
*/
```

© 2000-2025, MetaQuotes Ltd.

# CumProd

返回矩阵/向量元素的累积乘积，包括沿给定轴的累积乘积。

```
vector vector::CumProd();

vector matrix::CumProd();

matrix matrix::CumProd(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —每列的水平轴（即，整行），1 —每行的垂直轴（即，整列）

返回值

元素沿给定轴的累积乘积。

举例

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   matrix cols_cumprod=matrix_a.CumProd(0);
   matrix rows_cumprod=matrix_a.CumProd(1);
   vector cumprod_values=matrix_a.CumProd();

   Print("cols_cumprod\n",cols_cumprod);
   Print("rows_cumprod\n",rows_cumprod);
   Print("cumprod values  ",cumprod_values);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_cumprod
   [[10,3,2]
    [10,24,24]
    [60,120,96]
    [420,1320,864]]
   rows_cumprod
   [[10,30,60]
    [1,8,96]
    [6,30,120]
```

© 2000-2025, MetaQuotes Ltd.

```
    [7,77,693]]
   cumprod values  [10,30,60,60,480,5760,34560,172800,691200,4838400,53222400,47900160
   */
```

© 2000-2025, MetaQuotes Ltd.

# Percentile

返回矩阵/矢量元素或沿指定轴的元素值的指定百分位数。

```
double vector::Percentile(
  const int  percent      //
   );

double matrix::Percentile(
  const int  percent      //
   );

vector matrix::Percentile(
  const int  percent,     //
  const int  axis         // axis
   );
```

参数

*percent*

[输入] 欲计算的百分位数，必须介于 0 和 100 (含)之间。

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

百分位数：标量或向量。

注意

"percent" 参数的有效值范围在 [0, 100] 之内。采用线性算法计算百分位数。正确计算中位数需要对序列进行排序。

举例

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_percentile=matrix_a.Percentile(50,0);
   vectorf rows_percentile=matrix_a.Percentile(50,1);
   float matrix_percentile=matrix_a.Percentile(50);

   Print("cols_percentile ",cols_percentile);
   Print("rows_percentile ",rows_percentile);
   Print("percentile value  ",matrix_percentile);


   /*
   matrix_a
```

© 2000-2025, MetaQuotes Ltd.

```
   [[1,2,3]
    [4,5,6]
    [7,8,9]
    [10,11,12]]
   cols_percentile [5.5,6.5,7.5]
   rows_percentile [2,5,8,11]
   percentile value  6.5
   */
```

© 2000-2025, MetaQuotes Ltd.

# Quantile

返回矩阵/矢量元素或沿指定轴元素值的分位数。

```
double vector::Quantile(
  const double quantile      // quantile
   );

double matrix::Quantile(
  const double quantile      // quantile
   );

vector matrix::Quantile(
  const double quantile,     // quantile
  const int    axis          // axis
   );
```

参数

*quantile*

[输入] 欲计算的分位数，必须介于 0 和 1 (含)之间。

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

分位数：标量或向量。

注意

'quantile' 参数取值范围 [0, 1]。采用线性算法计算分位数。正确计算分位数需要对序列进行排序。

举例

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_quantile=matrix_a.Quantile(0.5,0);
   vectorf rows_quantile=matrix_a.Quantile(0.5,1);
   float matrix_quantile=matrix_a.Quantile(0.5);

   Print("cols_quantile ",cols_quantile);
   Print("rows_quantile ",rows_quantile);
   Print("quantile value  ",matrix_quantile);

   /*
   matrix_a
   [[1,2,3]
```

© 2000-2025, MetaQuotes Ltd.

```
    [4,5,6]
    [7,8,9]
    [10,11,12]]
   cols_quantile [5.5,6.5,7.5]
   rows_quantile [2,5,8,11]
   quantile value  6.5
   */
```

© 2000-2025, MetaQuotes Ltd.

# Median

计算矩阵/矢量元素的中位数。

```
double vector::Median();

double matrix::Median();

vector matrix::Median(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

中位数：标量或矢量。

注意

中位数是将数组/向量元素的最高一半与元素的最低一半分开的中间值。 与分位数 0.5)和百分位数 50)相同。正确计算中位数需要对序列进行排序。

举例

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_median=matrix_a.Median(0);
   vectorf rows_median=matrix_a.Median(1);
   float matrix_median=matrix_a.Median();

   Print("cols_median ",cols_median);
   Print("rows_median ",rows_median);
   Print("median value  ",matrix_median);


   /*
   matrix_a
   [[1,2,3]
    [4,5,6]
    [7,8,9]
    [10,11,12]]
   cols_median [5.5,6.5,7.5]
   rows_median [2,5,8,11]
```

© 2000-2025, MetaQuotes Ltd.

```
   median value  6.5
   */
```

© 2000-2025, MetaQuotes Ltd.

# Mean

计算元素值的算术平均值。

```
double vector::Mean();

double matrix::Mean();

vector matrix::Mean(
  const int  axis      // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

元素值的算术平均值。

举例

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_mean=matrix_a.Mean(0);
   vectorf rows_mean=matrix_a.Mean(1);
   float matrix_mean=matrix_a.Mean();

   Print("cols_mean ",cols_mean);
   Print("rows_mean ",rows_mean);
   Print("mean value  ",matrix_mean);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_mean [6,6.75,6.75]
   rows_mean [5,7,5,9]
   mean value  6.5
   */
```

© 2000-2025, MetaQuotes Ltd.

# Average

计算矩阵/向量值的加权平均值。

```
double vector::Average(
  const vector& weigts      // weights vector
   );

double matrix::Average(
  const matrix& weigts      // weights matrix
);

vector matrix::Average(
  const matrix& weigts,     // weights matrix
  const int     axis        // axis
   );
```

参数

*axis*

[输入] 轴。0 —水平轴，1 —垂直轴。

返回值

算术平均值：标量或向量。

注意

权重矩阵/向量与主矩阵/向量相关联。

举例

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   matrixf matrix_w=matrixf::Ones(4,3);
   Print("matrix_a\n",matrix_a);

   vectorf cols_average=matrix_a.Average(matrix_w,0);
   vectorf rows_average=matrix_a.Average(matrix_w,1);
   float matrix_average=matrix_a.Average(matrix_w);

   Print("cols_average ",cols_average);
   Print("rows_average ",rows_average);
   Print("average value  ",matrix_average);


   /*
   matrix_a
   [[10,3,2]
```

© 2000-2025, MetaQuotes Ltd.

```
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_average [6,6.75,6.75]
   rows_average [5,7,5,9]
   average value  6.5
   */ value  6.5
```

© 2000-2025, MetaQuotes Ltd.

# Std

返回矩阵/向量元素或沿给定轴的元素值的标准偏差。

```
double vector::Std();

double vector::Std(
  const int  ddof      // delta 自由度
);

double matrix::Std();

vector matrix::Std(
  const int  axis      // 轴
);

vector matrix::Std(
  const int  axis,     // 轴
  const int  ddof      // delta 自由度
);
```

参数

*axis*

[in] 轴。0 —水平轴，1 —垂直轴。

*ddof*

[in] “delta 自由度”：计算中使用的除数是 N-ddof，其中 N 表示元素的数量。默认情况下，ddof 为零。

返回值

标准偏差：标量或向量。

请注意

标准偏差是平均值平方偏差的平均值的平方根，即 std = sqrt(mean(x))，其中 x = abs(a - a.mean())**2.

平均平方偏差通常计算为 x.sum() / (N-ddof)，其中 N=len(x)。

ddof = 0 的表达式有时被称为“总体标准偏差”。如果 ddof > 0（最常用的是1），则所得量有时称为“样本标准偏差”。

示例

```
matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
Print("matrix_a\n",matrix_a);

vectorf cols_std=matrix_a.Std(0);
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf rows_std=matrix_a.Std(1);
   float matrix_std=matrix_a.Std();

   Print("cols_std ",cols_std);
   Print("rows_std ",rows_std);
   Print("std value  ",matrix_std);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_std [3.2403703,3.0310888,3.9607449]
   rows_std [3.5590262,4.5460606,0.81649661,1.6329932]
   std value  3.452052593231201
   */
```

© 2000-2025, MetaQuotes Ltd.

# Var

计算矩阵/向量元素值的方差。

```
double vector::Var();

double vector::Var(
  const int  ddof      // delta 自由度
);

double matrix::Var();

vector matrix::Var(
  const int  axis      // 轴
);

vector matrix::Var(
  const int  axis,     // 轴
  const int  ddof      // delta 自由度
);
```

参数

*axis*

[in] 轴。0 —水平轴，1 —垂直轴。

*ddof*

[in] “delta 自由度”：计算中使用的除数是 N-ddof，其中 N 表示元素的数量。默认情况下，ddof 为零。

返回值

方差：标量或向量。

请注意

方差是平均值的平方偏差的平均值，即 var=mean(x)，其中 x=abs(a-a.mean())**2。

平均值通常计算为 x.sum() / (N-ddof)，其中 N=len(x)。

ddof = 0 的表达式有时被称为“总体方差”。如果 ddof > 0（最常用的是1），则结果量有时称为“样本方差”。

示例

```
matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
Print("matrix_a\n",matrix_a);

vectorf cols_var=matrix_a.Var(0);
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf rows_var=matrix_a.Var(1);
   float matrix_var=matrix_a.Var();

   Print("cols_var ",cols_var);
   Print("rows_var ",rows_var);
   Print("var value  ",matrix_var);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_var [10.5,9.1875,15.6875]
   rows_var [12.666667,20.666666,0.66666669,2.6666667]
   var value  11.916666984558105
   */
```

© 2000-2025, MetaQuotes Ltd.

# LinearRegression

计算包含计算的线性回归值的向量/矩阵。

```
vector vector::LinearRegression();

matrix matrix::LinearRegression(
  ENUM_MATRIX_AXIS  axis=AXIS_NONE       // 沿其计算回归的轴线
   );
```

参数

*axis*

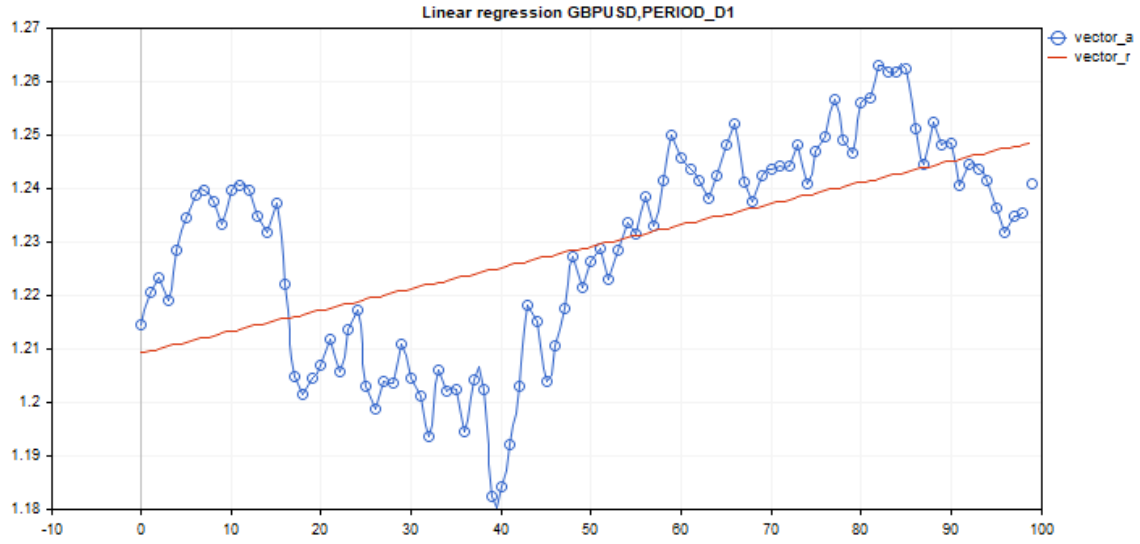
[in] 指定沿其计算回归的轴线。ENUM_MATRIX_AXIS枚举值(AXIS_HORZ —横轴，AXIS_VERT —竖轴)。

返回值

包含计算的线性回归值的向量/矩阵。

注意

使用标准回归方程计算线性回归：y (x) = a * x + b，其中a是直线斜率，而b是其Y轴位移。




示例：

```
#include <Graphics\Graphic.mqh>

#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnStart()
  {
   vector vector_a;
   vector_a.CopyRates(_Symbol,_Period,COPY_RATES_CLOSE,1,100);
   vector vector_r=vector_a.LinearRegression();

//--- 关闭图表显示
   ChartSetInteger(0,CHART_SHOW,false);

//--- 用于绘制图形的数组
   double x[];
   double y1[];
   double y2[];
   ArrayResize(x,uint(vector_a.Size()));
   ArrayResize(y1,uint(vector_a.Size()));
   ArrayResize(y2,uint(vector_a.Size()));
   for(ulong i=0; i<vector_a.Size(); i++)
     {
      x[i]=(double)i;
      y1[i]=vector_a[i];
      y2[i]=vector_r[i];
     }

//--- 图形标题
   string title="Linear regression "+_Symbol+","+EnumToString(_Period);

   long   chart=0;
   string name="LinearRegression";

//--- 创建图形
   CGraphic graphic;
   graphic.Create(chart,name,0,0,0,GRAPH_WIDTH,GRAPH_HEIGHT);
   graphic.BackgroundMain(title);
   graphic.BackgroundMainSize(12);

//--- 激活函数图
   CCurve *curvef=graphic.CurveAdd(x,y1,CURVE_POINTS_AND_LINES);
   curvef.Name("vector_a");
   curvef.LinesWidth(2);
   curvef.LinesSmooth(true);
   curvef.LinesSmoothTension(1);
   curvef.LinesSmoothStep(10);

//--- 激活函数的导数
   CCurve *curved=graphic.CurveAdd(x,y2,CURVE_LINES);
   curved.Name("vector_r");
   curved.LinesWidth(2);
   curved.LinesSmooth(true);
```

© 2000-2025, MetaQuotes Ltd.

```
   curved.LinesSmoothTension(1);
   curved.LinesSmoothStep(10);
   graphic.CurvePlotAll();
   graphic.Update();

//--- 无限循环识别按下的键盘按键
   while(!IsStopped())
     {
      //--- 按下escape键退出程序
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
         break;
      //--- 按下PdDn保存图形图片
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0)
        {
         string file_names[];
         if(FileSelectDialog("Save Picture",NULL,"All files (*.*)|*.*",FSD_WRITE_FILE,
            continue;
         ChartScreenShot(0,file_names[0],GRAPH_WIDTH,GRAPH_HEIGHT);
        }
      Sleep(10);
     }

//--- 清除
   graphic.Destroy();
   ObjectDelete(chart,name);
   ChartSetInteger(0,CHART_SHOW,true);
  }
```

### ENUM_MATRIX_AXIS

用于在矩阵所有统计函数中指定轴线的枚举。

| ID | 描述 |
| --- | --- |
| AXIS_NONE | 没有指定坐标轴。对所有矩阵元素执行计算，类似向量(请参见Flat方法)。 |
| AXIS_HORZ | 横轴 |
| AXIS_VERT | 竖轴 |

© 2000-2025, MetaQuotes Ltd.

# 特征方法

这些方法可以接收矩阵特征，诸如：

- 行数
- 列数
- 范数
- 条件数字
- 行列式
- 矩阵的秩
- 迹线
- 谱

| 函数 | 动作 |
| --- | --- |
| Rows | 返回矩阵中的行数 |
| Cols | 返回矩阵中的列数 |
| Size | 返回向量的大小 |
| Norm | 返回矩阵或向量范数 |
| Cond | 计算一个矩阵的条件数 |
| Det | 计算平方可逆矩阵的行列式 |
| SLogDet | 计算矩阵行列式的符号和对数 |
| Rank | 返回利用高斯（Gaussian）方法的矩阵秩 |
| Trace | 返回矩阵对角线的总和 |
| Spectrum | 计算矩阵的谱作为其本征值的集合，来自乘积 AT*A |

© 2000-2025, MetaQuotes Ltd.

# Rows

返回矩阵中的行数。

```
ulong matrix::Rows()
```

返回值

整数。

举例

```
matrix m= {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}};
m.Reshape(3, 4);
Print("matrix m \n" , m);
Print("m.Rows()=", m.Rows());
Print("m.Cols()=", m.Cols());

/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 m.Rows()=3
 m.Cols()=4
*/
```

© 2000-2025, MetaQuotes Ltd.

# Cols

返回矩阵中的列数。

```
ulong matrix::Cols()
```

返回值

整数。

举例

```
matrix m= {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}};
m.Reshape(3, 4);
Print("matrix m \n" , m);
Print("m.Cols()=", m.Cols());
Print("m.Rows()=", m.Rows());

/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 m.Cols()=4
 m.Rows()=3
*/
```

© 2000-2025, MetaQuotes Ltd.

## Size

返回向量的大小。

```
ulong vector::Size()
```

返回值

整数。

举例

```
matrix m={{1,2,3,4,5,6,7,8,9,10,11,12}};
m.Reshape(3,4);
Print("matrix m\n",m);
vector v=m.Row(1);
Print("v.Size()=",v.Size());
Print("v=",v);


/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 v.Size()=4
 v=[5,6,7,8]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Norm

返回矩阵或向量的范数。

```
double vector::Norm(
  const ENUM_VECTOR_NORM  norm,      // 向量范数
  const int               norm_p=2  // p-范数，如果是 VECTOR_NORM_P
   );

double matrix::Norm(
  const ENUM_MATRIX_NORM  norm       // 矩阵范数
   );
```

参数

*范数*

[输入] 范数顺序

返回值

矩阵或向量的范数。

注意

- VECTOR_NORM_INF 是向量元素中的最大绝对值。
- VECTOR_NORM_MINUS_INF 是向量元素中的最小绝对值。
- VECTOR_NORM_P 是向量的 P-范数。 如果 norm_p=0，那么这是非零向量元素的数量。norm_p=1 是矢量元素绝对值的总和。norm_p=2 是向量元素值平方和的平方根。P-范数可以为负值。
- MATRIX_NORM_FROBENIUS 是矩阵元素值的平方和的平方根。弗罗贝尼乌斯 (Frobenius) 范数和向量 P2-范数是一致的。
- MATRIX_NORM_SPECTRAL 是矩阵谱的最大值。
- MATRIX_NORM_NUCLEAR 是矩阵奇异值的总和。
- MATRIX_NORM_INF 是矩阵垂直向量中的最大向量 p1-范数。矩阵 inf-范数和向量 inf-范数是一致的。
- MATRIX_NORM_MINUS_INF 是矩阵垂直向量中的最小向量 p1-范数。
- MATRIX_NORM_P1 是矩阵水平向量中的最大向量 p1-范数。
- MATRIX_NORM_MINUS_P1 是矩阵水平向量中的最小向量 p1-范数。
- MATRIX_NORM_P2 是矩阵的最高奇异值。
- MATRIX_NORM_MINUS_P2 是矩阵的最低奇异值。

以 MQL5 实现的计算向量 P-范数的简单算法：

```
double VectorNormP(const vector& v,int norm_value)
  {
   ulong  i;
   double norm=0.0;
```

© 2000-2025, MetaQuotes Ltd.

```
//---
   switch(norm_value)
     {
      case 0 :
         for(i=0; i<v.Size(); i++)
            if(v[i]!=0)
               norm+=1.0;
         break;
      case 1 :
         for(i=0; i<v.Size(); i++)
            norm+=MathAbs(v[i]);
         break;
      case 2 :
         for(i=0; i<v.Size(); i++)
            norm+=v[i]*v[i];
         norm=MathSqrt(norm);
         break;
      default :
         for(i=0; i<v.Size(); i++)
            norm+=MathPow(MathAbs(v[i]),norm_value);
         norm=MathPow(norm,1.0/norm_value);
     }
//---
   return(norm);
  }
```

MQL5 示例：

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a=a-4;
  Print("matrix a \n", a);
  a.Reshape(3, 3);
  matrix b=a;
  Print("matrix b \n", b);
  Print("b.Norm(MATRIX_NORM_P2)=", b.Norm(MATRIX_NORM_FROBENIUS));
  Print("b.Norm(MATRIX_NORM_FROBENIUS)=", b.Norm(MATRIX_NORM_FROBENIUS));
  Print("b.Norm(MATRIX_NORM_INF)", b.Norm(MATRIX_NORM_INF));
  Print("b.Norm(MATRIX_NORM_MINUS_INF)", b.Norm(MATRIX_NORM_MINUS_INF));
  Print("b.Norm(MATRIX_NORM_P1)=)", b.Norm(MATRIX_NORM_P1));
  Print("b.Norm(MATRIX_NORM_MINUS_P1)=", b.Norm(MATRIX_NORM_MINUS_P1));
  Print("b.Norm(MATRIX_NORM_P2)=", b.Norm(MATRIX_NORM_P2));
  Print("b.Norm(MATRIX_NORM_MINUS_P2)=", b.Norm(MATRIX_NORM_MINUS_P2));

  /*
  matrix a
  [[-4,-3,-2,-1,0,1,2,3,4]]
  matrix b
  [[-4,-3,-2]
  [-1,0,1]
```

© 2000-2025, MetaQuotes Ltd.

```
  [2,3,4]]
  b.Norm(MATRIX_NORM_P2)=7.745966692414834
  b.Norm(MATRIX_NORM_FROBENIUS)=7.745966692414834
  b.Norm(MATRIX_NORM_INF)9.0
  b.Norm(MATRIX_NORM_MINUS_INF)2.0
  b.Norm(MATRIX_NORM_P1)=)7.0
  b.Norm(MATRIX_NORM_MINUS_P1)=6.0
  b.Norm(MATRIX_NORM_P2)=7.348469228349533
  b.Norm(MATRIX_NORM_MINUS_P2)=1.857033188519056e-16
  */
```

Python 示例：

```
import numpy as np
from numpy import linalg as LA
a = np.arange(9) - 4
print("a \n",a)
b = a.reshape((3, 3))
print("b \n",b)
print("LA.norm(b)=",LA.norm(b))
print("LA.norm(b, 'fro')=",LA.norm(b, 'fro'))
print("LA.norm(b, np.inf)=",LA.norm(b, np.inf))
print("LA.norm(b, -np.inf)=",LA.norm(b, -np.inf))
print("LA.norm(b, 1)=",LA.norm(b, 1))
print("LA.norm(b, -1)=",LA.norm(b, -1))
print("LA.norm(b, 2)=",LA.norm(b, 2))
print("LA.norm(b, -2)=",LA.norm(b, -2))

a
 [-4 -3 -2 -1  0  1  2  3  4]
b
 [[-4 -3 -2]
 [-1  0  1]
 [ 2  3  4]]
LA.norm(b)= 7.745966692414834
LA.norm(b, 'fro')= 7.745966692414834
LA.norm(b, np.inf)= 9.0
LA.norm(b, -np.inf)= 2.0
LA.norm(b, 1)= 7.0
LA.norm(b, -1)= 6.0
LA.norm(b, 2)= 7.3484692283495345
LA.norm(b, -2)= 1.857033188519056e-16
```

© 2000-2025, MetaQuotes Ltd.

# Cond

计算矩阵的条件数。

```
double matrix::Cond(
  const ENUM_MATRIX_NORM  norm      // 矩阵范数
   );
```

参数

*范数*

[输入] 来自 ENUM_MATRIX_NORM 的范数顺序

返回值

矩阵的条件数字。也许是无限的。

注意

x 的条件数定义为 x 的范数乘以 x 的逆范数 [1]。范数可以是通常的 L2-范数（平方和的根），或许多其它矩阵范数之一。

条件数 K 值，等于矩阵 A 的范数及其逆的乘积。条件数太高的矩阵称为病态。条件数较低的称为良态。逆矩阵采用伪反演得到，不受矩阵的直点和非奇点条件的限制。

谱条件数是一个例外。

以 MQL5 实现的计算谱条件数的简单算法：

```
double MatrixCondSpectral(matrix& a)
  {
   double norm=0.0;
   vector v=a.Spectrum();

   if(v.Size()>0)
     {
      double max_norm=v[0];
      double min_norm=v[0];
      for(ulong i=1; i<v.Size(); i++)
        {
         double real=MathAbs(v[i]);
         if(max_norm<real)
            max_norm=real;
         if(min_norm>real)
            min_norm=real;
        }
      max_norm=MathSqrt(max_norm);
      min_norm=MathSqrt(min_norm);
      if(min_norm>0.0)
```

© 2000-2025, MetaQuotes Ltd.

```
         norm=max_norm/min_norm;
     }

   return(norm);
  }
```

MQL5 示例：

```
  matrix a= {{1, 0, -1}, {0, 1, 0}, { 1, 0, 1}};
  Print("a.Cond(MATRIX_NORM_P2)=", a.Cond(MATRIX_NORM_P2));
  Print("a.Cond(MATRIX_NORM_FROBENIUS)=", a.Cond(MATRIX_NORM_FROBENIUS));
  Print("a.Cond(MATRIX_NORM_INF)=", a.Cond(MATRIX_NORM_INF));
  Print("a.Cond(MATRIX_NORM_MINUS_INF)=", a.Cond(MATRIX_NORM_MINUS_INF));
  Print("a.Cond(MATRIX_NORM_P1)=)", a.Cond(MATRIX_NORM_P1));
  Print("a.Cond(MATRIX_NORM_MINUS_P1)=", a.Cond(MATRIX_NORM_MINUS_P1));
  Print("a.Cond(MATRIX_NORM_P2)=", a.Cond(MATRIX_NORM_P2));
  Print("a.Cond(MATRIX_NORM_MINUS_P2)=", a.Cond(MATRIX_NORM_MINUS_P2));

  /*
  matrix a
  [[1,0,-1]
  [0,1,0]
  [1,0,1]]
  a.Cond(MATRIX_NORM_P2)=1.414213562373095
  a.Cond(MATRIX_NORM_FROBENIUS)=3.162277660168379
  a.Cond(MATRIX_NORM_INF)=2.0
  a.Cond(MATRIX_NORM_MINUS_INF)=0.9999999999999997
  a.Cond(MATRIX_NORM_P1)=)2.0
  a.Cond(MATRIX_NORM_MINUS_P1)=0.9999999999999998
  a.Cond(MATRIX_NORM_P2)=1.414213562373095
  a.Cond(MATRIX_NORM_MINUS_P2)=0.7071067811865472
  */
```

Python 示例：

```
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 0, -1], [0, 1, 0], [1, 0, 1]])
print("a \n",a)
print("LA.cond(a)=",LA.cond(a))
print("LA.cond(a, 'fro')=",LA.cond(a, 'fro'))
print("LA.cond(a, np.inf)=",LA.cond(a, np.inf))
print("LA.cond(a, -np.inf)=",LA.cond(a, -np.inf))
print("LA.cond(a, 1)=",LA.cond(a, 1))
print("LA.cond(a, -1)=",LA.cond(a, -1))
print("LA.cond(a, 2)=",LA.cond(a, 2))
print("LA.cond(a, -2)=",LA.cond(a, -2))
```

© 2000-2025, MetaQuotes Ltd.

```
a
 [[ 1  0 -1]
 [ 0  1  0]
 [ 1  0  1]]
LA.cond(a)= 1.4142135623730951
LA.cond(a, 'fro')= 3.1622776601683795
LA.cond(a, np.inf)= 2.0
LA.cond(a, -np.inf)= 1.0
LA.cond(a, 1)= 2.0
LA.cond(a, -1)= 1.0
LA.cond(a, 2)= 1.4142135623730951
LA.cond(a, -2)= 0.7071067811865475
```

© 2000-2025, MetaQuotes Ltd.

# Det

计算正方形可逆矩阵的行列式。

```
double matrix::Det()
```

返回值

矩阵的行列式。

注意

二阶和三阶矩阵行列式根据萨鲁斯（Sarrus）规则进行计算。 d2=a11*a22-a12*a21; d3=a11*a22*a33+a12*a23*a31+a13*a21*a32-a13*a22*a31-a11*a23*a32-a12*a21*a33

行列式是利用高斯（Gaussian）方法将矩阵简化为上三角形计算出的。上三角矩阵的行列式等于主对角线元素的乘积。

如果至少一个矩阵的行或列为零，则行列式为零。

如果两个或多个矩阵的行或列线性相关，则其行列式为零。

矩阵的行列式等于其本征值的乘积。

MQL5 示例：

```
   matrix m={{1,2},{3,4}};
   double det=m.Det();
   Print("matrix m\n",m);
   Print("det(m)=",det);
   /*
   matrix m
   [[1,2]
    [3,4]]
   det(m)=-2.0
   */
```

Python 示例：

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
print('a \n',a)
print('nnp.linalg.det(a) \n',np.linalg.det(a))

a
 [[1 2]
 [3 4]]
```

© 2000-2025, MetaQuotes Ltd.

```
np.linalg.det(a)
 -2.0000000000000004
```

© 2000-2025, MetaQuotes Ltd.

# SLogDet

计算矩阵行列式的符号和对数。

```
double matrix::SLogDet(
  int&  sign      // 符号
   );
```

参数

*sign*

[输出]行列式的符号。如果符号为偶数，则行列式为正数。

返回值

表示行列式符号的数字。

注意

行列式是利用高斯 Gaussian)方法将矩阵简化为上三角形计算出的。上三角矩阵的行列式等于主对角线元素的乘积。乘积的对数等于对数之和。因此，如果在计算行列式时出现溢出，则可以使用 SLogDet 方法。

如果符号为偶数，则行列式为正数。

举例

```
a = np.array([[1, 2], [3, 4]])
(sign, logdet) = np.linalg.slogdet(a)
(sign, logdet) (-1, 0.69314718055994529) # may vary  sign * np.exp(logdet) -2.0
```

© 2000-2025, MetaQuotes Ltd.

# Rank

利用高斯（Gaussian）方法返回矩阵的秩。

```
int  Rank()
```

返回值

矩阵的秩。

注意

具有 m 行和 n 列的矩阵 A 的行（或列）系统的秩是线性独立行（或列）的最大数目。如果若干行（列）中无一可用其它行（列）线性表达，则称为线性独立。 行系统的秩始终等于列系统的秩。此值称为矩阵的秩。

MQL5 示例：

```
matrix a=matrix::Eye(4, 4);;
Print("matrix a \n", a);
Print("a.Rank()=", a.Rank());

matrix I=matrix::Eye(4, 4);
I[3, 3] = 0.;    // 无秩矩阵
Print("I \n", I);
Print("I.Rank()=", I.Rank());

matrix b=matrix::Ones(1, 4);
Print("b \n", b);
Print("b.Rank()=", b.Rank());;// 1 维 - 秩为 1 除外所有 0

matrix  zeros=matrix::Zeros(4, 1);
Print("zeros \n", zeros);
Print("zeros.Rank()=", zeros.Rank());

/*
matrix a
[[1,0,0,0]
[0,1,0,0]
[0,0,1,0]
[0,0,0,1]]
a.Rank()=4

I
[[1,0,0,0]
[0,1,0,0]
[0,0,1,0]
[0,0,0,0]]
```

© 2000-2025, MetaQuotes Ltd.

```
  I.Rank()=3

  b
  [[1,1,1,1]]
  b.Rank()=1

  zeros
  [[0]
  [0]
  [0]
  [0]]
  zeros.Rank()=0
  */
```

Python 示例：

```
import numpy as np
from numpy.linalg import matrix_rank
a=(np.eye(4)) # 全 秩 矩阵
print("a \n", a)
print("matrix_rank(a)=",matrix_rank(a))
I=np.eye(4)
I[-1,-1] = 0. # 秩 缺乏 矩阵x
print("I \n",I)
print("matrix_rank(I)=",matrix_rank(I))

b=np.ones((4,))
print("b \n",b)
print("matrix_rank(b)=",matrix_rank(b)) # 1 维 - 秩为 1 除外 全部 0

zeros=np.zeros((4,))
print("zeroes \n",zeros)
print("matrix_rank(zeros)=",matrix_rank(zeros))

a
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
matrix_rank(a)= 4

I
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
matrix_rank(I)= 3
```

© 2000-2025, MetaQuotes Ltd.

```
b
 [1. 1. 1. 1.]
matrix_rank(b)= 1

zeroes
 [0. 0. 0. 0.]
matrix_rank(zeros)= 0
```

© 2000-2025, MetaQuotes Ltd.

# Trace

返回沿矩阵对角线的总和。

```
double matrix::Trace()
```

返回值

沿对角线的总和。

注意

矩阵的迹线等于其本征值的总和。

MQL5 示例：

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a.Reshape(3, 3);
  Print("matrix a \n", a);
  Print("a.Trace() \n", a.Trace());

  /*
  matrix a
  [[0,1,2]
  [3,4,5]
  [6,7,8]]
  a.Trace()
  12.0
  */
```

Python 示例：

```
a = np.arange(9).reshape((3,3))
print('a \n',a)
print('np.trace(a) \n',np.trace(a))

a
 [[0 1 2]
 [3 4 5]
 [6 7 8]]

np.trace(a)
 12
```

© 2000-2025, MetaQuotes Ltd.

# Spectrum

计算矩阵的谱，作为来自乘积 AT*A 的本征值集合。

```
vector matrix::Spectrum()
```

返回值

矩阵的谱作为矩阵本征值的向量。

举例

```
double MatrixCondSpectral(matrix& a)
  {
   double norm=0.0;
   vector v=a.Spectrum();

   if(v.Size()>0)
     {
      double max_norm=v[0];
      double min_norm=v[0];
      for(ulong i=1; i<v.Size(); i++)
        {
         double real=MathAbs(v[i]);
         if(max_norm<real)
            max_norm=real;
         if(min_norm>real)
            min_norm=real;
        }
      max_norm=MathSqrt(max_norm);
      min_norm=MathSqrt(min_norm);
      if(min_norm>0.0)
         norm=max_norm/min_norm;
     }

   return(norm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 求解线性方程组的矩阵方法

求解线性方程组和计算逆矩阵的方法。

| 函数 | 动作 |
| --- | --- |
| Solve | 求解线性矩阵方程或线性代数方程组 |
| LstSq | 返回线性代数方程的最小二乘解（对于非二乘或退化矩阵） |
| Inv | 通过乔丹-高斯（Jordan-Gauss）方法计算平方非退化矩阵的（乘法）逆 |
| PInv | 遵照摩尔-彭罗斯（Moore-Penrose）方法计算矩阵的伪逆 |

© 2000-2025, MetaQuotes Ltd.

# Solve

求解线性矩阵方程或线性代数方程组。

```
vector matrix::Solve(
  const vector  b      // 纵坐标或"因变量"值
   );
```

参数

*b*

[输入] 纵坐标或"因变量"值。（自由项的向量）。

返回值

含有系统 a*x = b 之解的向量。

注意

如果至少一个矩阵行或列为零，则系统无解。

如果两个或多个矩阵行或列线性依赖，则系统无解。

举例

```
//--- SLAE 解法
   vector_x=matrix_a.Solve(vector_b);
//--- 检查是否 a * x = b
   result_vector=matrix_a.MatMul(vector_x);
   errors=vector_b.Compare(result_vector,1e-12);
```

© 2000-2025, MetaQuotes Ltd.

# LstSq

返回线性代数方程的最小二乘解（对于非二乘或退化矩阵）。

```
vector matrix::LstSq(
  const vector  b      // 纵坐标或"因变量"值
   );
```

参数

*b*

[输入] 纵坐标或"因变量"值。（自由项的向量）

返回值

含有系统 a*x = b 之解的向量。这仅适用于具有精准解的系统。

举例

```
   matrix a={{3, 2},
             {4,-5},
             {3, 3}};
   vector b={7,40,3};
//---
   vector x=a.LstSq(b);
//--- 检查，必须是 [5, -4]
   Print("x=", x);
//--- 检查，必须是 [7, 40, 3]
   vector b1=a.MatMul(x);
   Print("b1=",b1);

/*
  x=[5.000000000000002,-4]
  b1=[7.000000000000005,40.00000000000001,3.000000000000005]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Inv

通过乔丹-高斯 (Jordan-Gauss)方法计算平方可逆矩阵的乘法逆。

```
matrix matrix::Inv()
```

返回值

矩阵的乘法逆。

注意

原始矩阵和逆矩阵的乘积就是单位矩阵。

如果至少一个矩阵行或列为零，则无法获得逆矩阵。

如果两个或多个矩阵行或列线性依赖，则无法获得逆矩阵。

举例

```
int TestInverse(const int size_m)
  {
   int    i,j,errors=0;
   matrix matrix_a(size_m,size_m);
//--- 填充平方矩阵
   MatrixTestFirst(matrix_a);
//--- 将测量单位微秒
   ulong t1=GetMicrosecondCount();
//--- 求逆矩阵
   matrix inverse=matrix_a.Inv();
//--- 测量
   ulong t2=GetMicrosecondCount();
//--- 检查正确性
   matrix identity=matrix_a.MatMul(inverse);
//---
   for(i=0; i<size_m; i++)
     {
      for(j=0; j<size_m; j++)
        {
         double value;
         //--- 必须沿着对角线填充一
         if(i==j)
            value=1.0;
         else
            value=0.0;
         if(MathClassify(identity[i][j])>FP_ZERO)
            errors++;
         else
```

© 2000-2025, MetaQuotes Ltd.

```
            {
             if(identity[i][j]!=value)
               {
                double diff=MathAbs(identity[i][j]-value);
                //--- 乘法和偏差过多，因此降低了检查精度
                if(diff>1e-9)
                   errors++;
               }
            }
         }
      }
//---
   double elapsed_time=double(t2-t1)/1000.0;
   printf("Inversion of matrix %d x %d  %s  errors=%d  time=%.3f ms",size_m,size_m,err
   return(errors);
  }
```

© 2000-2025, MetaQuotes Ltd.

# PInv

通过摩尔-彭罗斯（Moore-Penrose）方法计算矩阵的伪逆。

```
matrix matrix::PInv()
```

返回值

矩阵的伪逆。

举例

```
int TestPseudoInverse(const int size_m, const int size_k)
  {
   matrix matrix_a(size_m,size_k);
   matrix matrix_inverted(size_k,size_m);
   matrix matrix_temp;
   matrix matrix_a2;
//--- 填充矩阵
   MatrixTestFirst(matrix_a);
//--- invert
   matrix_inverted=matrix_a.PInv();
//--- 检查正确性
   int errors=0;
//--- A * A+ * A = A   (A+ is a pseudo-inverse of A)
   matrix_temp=matrix_a.MatMul(matrix_inverted);
   matrix_a2=matrix_temp.MatMul(matrix_a);
   errors=(int)matrix_a.CompareByDigits(matrix_a2,10);

   printf("PseudoInversion %s matrix_size  %d x %d  errors=%d",errors==0?"passed":"fai
//---
   return(errors);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 机器学习

这些方法用于机器学习。

神经网络激活函数根据输入的加权累加和来判定神经元的输出值。 激活函数的选择对神经网络性能有很大影响。模型部件（层）不同可选用不同的激活函数。

除了所有已知的激活函数外，MQL5 还提供其衍生函数。 函数导数能够根据学习中接到的误差有效地更新模型参数。

神经网络旨在找到一种算法，将学习误差最小化，为此选用了损失函数。损失函数的值表示模型预测值与实际值的偏差度。 根据问题采用不同的损失函数。 例如，均方误差（MSE）用于回归问题；二元交叉熵（BCE）用于二元分类目的。

| 函数 | 动作 |
| --- | --- |
| Activation | 计算激活函数值，并将其写入传递的向量/矩阵 |
| Derivative | 计算激活函数导数值，并将其写入所传递的向量/矩阵 |
| Loss | 计算损失函数值，并将其写入所传递的向量/矩阵 |
| LossGradient | 计算亏损函数梯度的向量或矩阵 |
| RegressionMetric | 在指定数据数组上构建回归线，并计算回归量程作为偏差误差 |
| ConfusionMatrix | 计算混淆矩阵。该方法应用于预测值向量 |
| ConfusionMatrixMultilabel | 计算每个标签的混淆矩阵。该方法应用于预测值向量 |
| ClassificationMetric | 计算分类指标来评估预测数据相对于真实数据的质量。该方法适用于预测值的向量 |
| ClassificationScore | 计算分类指标来评估预测数据相对于真实数据的质量 |
| PrecisionRecall | 计算值以构建精确调用曲线。与ClassificationScore类似，此方法应用于真值向量 |
| ReceiverOperatingCharacteristic | 计算值以构建接收器工作特性(ROC)曲线。与ClassificationScore类似，此方法应用于真值向量 |

举例

此示例演示如何选用矩阵运算训练模型。 训练的模型针对函数 (a + b + c)^2 / (a^2 + b^2 + c^2)。 我们输入初始数据矩阵，其中 a、b 和 c 包含在不同的列中。 函数结果在模型输出端获得。

```
matrix weights1, weights2, weights3;                    // 权重矩阵
matrix output1, output2, result;                        // 神经层输出矩阵
input int layer1 = 200;                                 // 第一个隐藏层的大小
input int layer2 = 200;                                 // 第二个隐藏层的大小
input int Epochs = 20000;                               // 训练世代的数量
input double lr = 3e-6;                                 // 学习率
input ENUM_ACTIVATION_FUNCTION ac_func = AF_SWISH; // 激活函数
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序 start 函数                                              |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int train = 1000;     // 训练样本大小
   int test = 10;        // 测试样本大小
   matrix m_data, m_target;
//--- 生成训练样本
   if(!CreateData(m_data, m_target, train))
      return;
//--- 训练模型
   if(!Train(m_data, m_target, Epochs))
      return;
//--- 生成测试样本
   if(!CreateData(m_data, m_target, test))
      return;
//--- 测试模型
   Test(m_data, m_target);
  }
//+------------------------------------------------------------------+
//| 样本生成方法                                                     |
//+------------------------------------------------------------------+
bool CreateData(matrix &data, matrix &target, const int count)
  {
//--- 初始化初始数据和结果矩阵
   if(!data.Init(count, 3) || !target.Init(count, 1))
      return false;
//--- 用随机值填充初始数据矩阵
   data.Random(-10, 10);
//--- 计算训练样本的目标值
   vector X1 = MathPow(data.Col(0) + data.Col(1) + data.Col(1), 2);
   vector X2 = MathPow(data.Col(0), 2) + MathPow(data.Col(1), 2) + MathPow(data.Col(2)
   if(!target.Col(X1 / X2, 0))
      return false;
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
//| 模型训练方法                                                     |
//+------------------------------------------------------------------+
bool Train(matrix &data, matrix &target, const int epochs = 10000)
  {
//--- 创建模型
   if(!CreateNet())
      return false;
//--- 训练模型
   for(int ep = 0; ep < epochs; ep++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 前向验算
       if(!FeedForward(data))
          return false;
       PrintFormat("Epoch %d, loss %.5f", ep, result.Loss(target, LOSS_MSE));
       //--- 权重矩阵的反向传播与更新
       if(!Backprop(data, target))
          return false;
      }
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
//| 模型创建方法                                                     |
//+------------------------------------------------------------------+
bool CreateNet()
  {
//--- 初始化权重矩阵
   if(!weights1.Init(4, layer1) || !weights2.Init(layer1 + 1, layer2) || !weights3.Ini
      return false;
//--- 用随机值填充权重矩阵
   weights1.Random(-0.1, 0.1);
   weights2.Random(-0.1, 0.1);
   weights3.Random(-0.1, 0.1);
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
//| 前向验算方法                                                     |
//+------------------------------------------------------------------+
bool FeedForward(matrix &data)
  {
//--- 检查初始数据大小
   if(data.Cols() != weights1.Rows() - 1)
      return false;
//--- 计算第一个神经层
   matrix temp = data;
   if(!temp.Resize(temp.Rows(), weights1.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights1.Rows() - 1))
      return false;
   output1 = temp.MatMul(weights1);
//--- 计算激活函数
   if(!output1.Activation(temp, ac_func))
      return false;
//--- 计算第二个神经层
   if(!temp.Resize(temp.Rows(), weights2.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights2.Rows() - 1))
      return false;
   output2 = temp.MatMul(weights2);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 计算激活函数
   if(!output2.Activation(temp, ac_func))
      return false;
//--- 计算第三个神经层
   if(!temp.Resize(temp.Rows(), weights3.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights3.Rows() - 1))
      return false;
   result = temp.MatMul(weights3);
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
//| 反向传播法                                                       |
//+------------------------------------------------------------------+
bool Backprop(matrix &data, matrix &target)
  {
//--- 检查目标值矩阵的大小
   if(target.Rows() != result.Rows() ||
      target.Cols() != result.Cols())
      return false;
//--- 检测计算值与目标的偏差
   matrix loss = (target - result) * 2;
//--- 将梯度传播到上一层
   matrix gradient = loss.MatMul(weights3.Transpose());
//--- 更新最后一层的权重矩阵
   matrix temp;
   if(!output2.Activation(temp, ac_func))
      return false;
   if(!temp.Resize(temp.Rows(), weights3.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights3.Rows() - 1))
      return false;
   weights3 = weights3 + temp.Transpose().MatMul(loss) * lr;
//--- 依据激活函数的导数调整误差梯度
   if(!output2.Derivative(temp, ac_func))
      return false;
   if(!gradient.Resize(gradient.Rows(), gradient.Cols() - 1))
      return false;
   loss = gradient * temp;
//--- 将梯度传播到更低一层
   gradient = loss.MatMul(weights2.Transpose());
//--- 更新第二隐藏层的权重矩阵
   if(!output1.Activation(temp, ac_func))
      return false;
   if(!temp.Resize(temp.Rows(), weights2.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights2.Rows() - 1))
      return false;
   weights2 = weights2 + temp.Transpose().MatMul(loss) * lr;
//--- 依据激活函数的导数调整误差梯度
   if(!output1.Derivative(temp, ac_func))
```

© 2000-2025, MetaQuotes Ltd.

```
      return false;
   if(!gradient.Resize(gradient.Rows(), gradient.Cols() - 1))
      return false;
   loss = gradient * temp;
//--- 更新第一隐藏层的权重矩阵
   temp = data;
   if(!temp.Resize(temp.Rows(), weights1.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights1.Rows() - 1))
      return false;
   weights1 = weights1 + temp.Transpose().MatMul(loss) * lr;
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
//| 模型测试方法                                                     |
//+------------------------------------------------------------------+
bool Test(matrix &data, matrix &target)
  {
//--- 前向测试数据
   if(!FeedForward(data))
      return false;
//--- 记录模型计算结果和真实值
   PrintFormat("Test loss %.5f", result.Loss(target, LOSS_MSE));
   ulong total = data.Rows();
   for(ulong i = 0; i < total; i++)
      PrintFormat("(%.2f + %.2f + %.2f)^2 / (%.2f^2 + %.2f^2 + %.2f^2) =  Net %.2f, Ta
                  data[i, 0], data[i, 1], data[i, 2], result[i, 0], target[i, 0]);
//--- 返回结果
   return true;
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# Activation

计算激活函数值，并将其写入传递的向量/矩阵。

```
bool vector::Activation(
  vector&                    vect_out,      // 获取函数值的向量
  ENUM_ACTIVATION_FUNCTION   activation,    // 激活函数
   ...                                      // 附加参数
   );


bool matrix::Activation(
  matrix&                    matrix_out,    // 获取函数值的矩阵
  ENUM_ACTIVATION_FUNCTION   activation     // 激活函数
   );


bool matrix::Activation(
  matrix&                    matrix_out,    // 获取函数值的矩阵
  ENUM_ACTIVATION_FUNCTION   activation,    // 激活函数
  ENUM_MATRIX_AXIS           axis,          // 轴线
   ...                                      // 附加参数
   );
```

参数

*vect_out/matrix_out*

[out] 获取激活函数计算值的向量或矩阵。

*activation*

[in] ENUM_ACTIVATION_FUNCTION枚举中的激活函数。

*axis*

[in] ENUM_MATRIX_AXIS枚举值（AXIS_HORZ —横轴，AXIS_VERT —竖轴）。

*...*

[in] 某些激活函数所需的附加参数。如果没有指定参数，则使用默认值。

返回值

如果成功返回true，否则返回false。

附加参数

某些激活函数接受附加参数。如果没有指定参数，则使用默认值

```
   AF_ELU   (Exponential Linear Unit)
```

© 2000-2025, MetaQuotes Ltd.

```
  double alpha=1.0

Activation function: if(x>=0) f(x) = x
                     else f(x) = alpha * (exp(x)-1)


AF_LINEAR
  double alpha=1.0
  double beta=0.0

Activation function: f(x) = alpha*x + beta


AF_LRELU   (Leaky REctified Linear Unit)
  double alpha=0.3

Activation function: if(x>=0) f(x)=x
                     else f(x) = alpha*x


AF_RELU  (REctified Linear Unit)
  double alpha=0.0
  double max_value=0.0
  double treshold=0.0

Activation function: if(alpha==0) f(x) = max(x,0)
                     else if(x>max_value) f(x) = x
                     else f(x) = alpha*(x - treshold)


AF_SWISH
  double beta=1.0

Activation function: f(x) = x / (1+exp(-x*beta))


AF_TRELU   (Thresholded REctified Linear Unit)
  double theta=1.0

Activation function: if(x>theta) f(x) = x
                     else f(x) = 0


AF_PRELU   (Parametric REctified Linear Unit)
  double alpha[] - learned array of coeefficients

Activation function: if(x[i]>=0) f(x)[i] = x[i]
                     else f(x)[i] = alpha[i] * x[i]
```

© 2000-2025, MetaQuotes Ltd.

注意

在人工神经网络中，神经元的激活函数决定输出信号，输出信号由一个输入信号或一组输入信号定义。激活函数的选择对神经网络的性能有很大的影响。不同的模型部件(层)可以使用不同的激活函数。

使用附加参数的示例：

```
vector x={0.1, 0.4, 0.9, 2.0, -5.0, 0.0, -0.1};
vector y;

x.Activation(y,AF_ELU);
Print(y);
x.Activation(y,AF_ELU,2.0);
Print(y);

Print("");
x.Activation(y,AF_LINEAR);
Print(y);
x.Activation(y,AF_LINEAR,2.0);
Print(y);
x.Activation(y,AF_LINEAR,2.0,5.0);
Print(y);

Print("");
x.Activation(y,AF_LRELU);
Print(y);
x.Activation(y,AF_LRELU,1.0);
Print(y);
x.Activation(y,AF_LRELU,0.1);
Print(y);

Print("");
x.Activation(y,AF_RELU);
Print(y);
x.Activation(y,AF_RELU,2.0,0.5);
Print(y);
x.Activation(y,AF_RELU,2.0,0.5,1.0);
Print(y);

Print("");
x.Activation(y,AF_SWISH);
Print(y);
x.Activation(y,AF_SWISH,2.0);
Print(y);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("");
   x.Activation(y,AF_TRELU);
   Print(y);
   x.Activation(y,AF_TRELU,0.3);
   Print(y);

   Print("");
   vector a=vector::Full(x.Size(),2.0);
   x.Activation(y,AF_PRELU,a);
   Print(y);

/*  Results
   [0.1,0.4,0.9,2,-0.993262053000915,0,-0.095162581964040]
   [0.1,0.4,0.9,2,-1.986524106001829,0,-0.190325163928081]

   [0.1,0.4,0.9,2,-5,0,-0.1]
   [0.2,0.8,1.8,4,-10,0,-0.2]
   [5.2,5.8,6.8,9,-5,5,4.8]

   [0.1,0.4,0.9,2,-1.5,0,-0.03]
   [0.1,0.4,0.9,2,-5,0,-0.1]
   [0.1,0.4,0.9,2,-0.5,0,-0.01]

   [0.1,0.4,0.9,2,0,0,0]
   [0.2,0.8,0.9,2,-10,0,-0.2]
   [-1.8,-1.2,0.9,2,-12,-2,-2.2]

   [0.052497918747894,0.239475064044981,0.639854552362503,1.761594155955765,-0.033464
   [0.054983399731247,0.275989792451045,0.772334041589561,1.964027580075817,-0.000226

   [0,0,0,2,0,0,0]
   [0,0.4,0.9,2,0,0,0]

   [0.1,0.4,0.9,2,-10,0,-0.2]
*/
```

© 2000-2025, MetaQuotes Ltd.

# Derivative

计算激活函数的导数值，并将其写入传递的向量/矩阵。

```
bool vector::Derivative(
  vector&                    vect_out,      // 获取函数值的向量
  ENUM_ACTIVATION_FUNCTION   activation,    // 激活函数
   ...                                      // 附加参数
   );


bool matrix::Derivative(
  matrix&                    matrix_out,    // 获取函数值的矩阵
  ENUM_ACTIVATION_FUNCTION   activation,    // 激活函数
   );


bool matrix::Derivative(
  matrix&                    matrix_out,    // 获取函数值的矩阵
  ENUM_ACTIVATION_FUNCTION   activation,    // 激活函数
  ENUM_MATRIX_AXIS           axis,          // 轴线
   ...                                      // 附加参数
   );
```

参数

*vect_out/matrix_out*

[out] 获取激活函数导数计算值的向量或矩阵。

*activation*

[in] ENUM_ACTIVATION_FUNCTION枚举中的激活函数。

*axis*

[in] ENUM_MATRIX_AXIS枚举值（AXIS_HORZ —横轴，AXIS_VERT —竖轴）。

*...*

[in] 附加参数与激活函数的参数相同。只有某些激活函数接受附加参数。如果没有指定参数，则使用默认值。

返回值

如果成功返回true，否则返回false。

注意

在误差反向传播过程中，函数导数能够基于学习过程中接收到的误差有效地更新模型参数。

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

## Loss

计算损失函数的值。

```
double vector::Loss(
  const vector&        vect_true,     // 真值向量
  ENUM_LOSS_FUNCTION   loss,          // 损失函数
   ...                                // 附加参数
   );


double matrix::Loss(
  const matrix&        matrix_true,   // 真值矩阵
  ENUM_LOSS_FUNCTION   loss,          // 损失函数
   );


double matrix::Loss(
  const matrix&        matrix_true,   // 真值矩阵
  ENUM_LOSS_FUNCTION   loss,          // 损失函数
  ENUM_MATRIX_AXIS     axis,          // 轴线
   ...                                // 附加参数
   );
```

参数

*vect_true/matrix_true*

[in] 真值的向量或矩阵。

*loss*

[in] ENUM_LOSS_FUNCTION枚举中的损失函数。

*axis*

[in] ENUM_MATRIX_AXIS枚举值（AXIS_HORZ—横轴，AXIS_VERT—竖轴）。

*...*

[in] 附加参数'delta'只能由Hubert损失函数(LOSS_HUBER)使用

返回值

双数值。

如果Hubert损失函数(LOSS_HUBER)中使用在'delta'参数

```
   double delta = 1.0;
   double error = fabs(y - x);
   if(error<delta)
      loss = 0.5 * error^2;
   else
      loss = 0.5 * delta^2 + delta * (error - delta);
```

© 2000-2025, MetaQuotes Ltd.

注意

神经网络的目标是找到最小化训练样本误差的算法，为此使用损失函数。

损失函数的值表示模型预测值与实际值偏差的程度。

根据不同的问题使用不同的损失函数。例如，均方误差(MSE)用于回归问题，二元交叉熵(BCE)用于二元分类。

调用Hubert损失函数的示例：

```
   vector y_true = {0.0, 1.0, 0.0, 0.0};
   vector y_pred = {0.6, 0.4, 0.4, 0.6};
   double loss=y_pred.Loss(y_true,LOSS_HUBER);
   Print(loss);
   double loss2=y_pred.Loss(y_true,LOSS_HUBER,0.5);
   Print(loss2);

/* Result
   0.155
   0.15125
*/
```

© 2000-2025, MetaQuotes Ltd.

# LossGradient

计算损失函数梯度的向量或矩阵。

```
vector vector::LossGradient(
  const vector&        vect_true,      // 真值向量
  ENUM_LOSS_FUNCTION   loss,           // 损失函数类型
   ...                                 // 附加参数
   );


matrix matrix::LossGradient(
  const matrix&        matrix_true,    // 真值矩阵
  ENUM_LOSS_FUNCTION   loss,           // 损失函数
   );


matrix matrix::LossGradient(
  const matrix&        matrix_true,    // 真值矩阵
  ENUM_LOSS_FUNCTION   loss,           // 损失函数
  ENUM_MATRIX_AXIS     axis,           // 轴线
   ...                                 // 附加参数
   );
```

参数

*vect_true/matrix_true*

[in] 真值的向量或矩阵。

*loss*

[in] ENUM_LOSS_FUNCTION枚举中的损失函数。

*axis*

[in] ENUM_MATRIX_AXIS枚举值（AXIS_HORZ —横轴，AXIS_VERT —竖轴）。

*...*

[in] 附加参数'delta'只能由Hubert损失函数(LOSS_HUBER)使用

返回值

损失函数梯度值的向量或矩阵。梯度是损失函数在给定点上对dx的偏导数（x是预测值）。

注意

神经网络在训练模型时，在反向传播过程中使用梯度来调整权重矩阵的权重。

神经网络的目标是找到最小化训练样本误差的算法，为此使用损失函数。

根据不同的问题使用不同的损失函数。例如，均方误差(MSE)用于回归问题，二元交叉熵(BCE)用于二元分类。

© 2000-2025, MetaQuotes Ltd.

计算损失函数梯度的示例

```
   matrixf y_true={{ 1, 2, 3, 4 },
                   { 5, 6, 7, 8 },
                   { 9,10,11,12 }};
   matrixf y_pred={{ 1, 2, 3, 4 },
                   {11,10, 9, 8 },
                   { 5, 6, 7,12 }};
   matrixf loss_gradient =y_pred.LossGradient(y_true,LOSS_MAE);
   matrixf loss_gradienth=y_pred.LossGradient(y_true,LOSS_MAE,AXIS_HORZ);
   matrixf loss_gradientv=y_pred.LossGradient(y_true,LOSS_MAE,AXIS_VERT);
   Print("loss gradients\n",loss_gradient);
   Print("loss gradients on horizontal axis\n",loss_gradienth);
   Print("loss gradients on vertical axis\n",loss_gradientv);

/* Result
   loss gradients
   [[0,0,0,0]
    [0.083333336,0.083333336,0.083333336,0]
    [-0.083333336,-0.083333336,-0.083333336,0]]
   loss gradients on horizontal axis
   [[0,0,0,0]
    [0.33333334,0.33333334,0.33333334,0]
    [-0.33333334,-0.33333334,-0.33333334,0]]
   loss gradients on vertical axis
   [[0,0,0,0]
    [0.25,0.25,0.25,0]
    [-0.25,-0.25,-0.25,0]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# RegressionMetric

计算回归指标来评估预测数据与真实数据相比的质量

```
double vector::RegressionMetric(
   const vector&            vector_true,   // 真值向量
   ENUM_REGRESSION_METRIC   metric         // 指标类型
   );

double matrix::RegressionMetric(
   const matrix&            matrix_true,   // 真值矩阵
   ENUM_REGRESSION_METRIC   metric         // 指标类型
);

vector matrix::RegressionMetric(
   const matrix&            matrix_true,   // 真值矩阵
   ENUM_REGRESSION_METRIC   metric,        // 指标类型
   int                      axis           // 轴线
   );
```

参数

*vector_true/matrix_true*

[in] 真值的向量或矩阵。

*metric*

[in] 来自ENUM_REGRESSION_METRIC枚举的指标类型。

*axis*

[in] 轴线。0 —横轴，1 —竖轴。

返回值

计算用于评估预测数据与真实数据相比的质量的指标

注意

- REGRESSION_MAE —平均绝对误差，表示预测值与相应真值之间的绝对差值
- REGRESSION_MSE —均方误差，表示预测值与对应真值之间的平方差
- REGRESSION_RMSE —均方根误差(RMSE)
- REGRESSION_R2 - 1 —MSE (回归)/ MSE (平均)
- REGRESSION_MAPE —平均绝对误差(MAE)百分比
- REGRESSION_MSPE —均方误差(MSE)百分比
- REGRESSION_RMSLE —按对数尺计算的RMSE (均方根误差)

例如：

```
   vector y_true = {3, -0.5, 2, 7};
   vector y_pred = {2.5, 0.0, 2, 8};
```

© 2000-2025, MetaQuotes Ltd.

```
//---
   double mse=y_pred.RegressionMetric(y_true,REGRESSION_MSE);
   Print("mse=",mse);
//---
   double mae=y_pred.RegressionMetric(y_true,REGRESSION_MAE);
   Print("mae=",mae);
//---
   double r2=y_pred.RegressionMetric(y_true,REGRESSION_R2);
   Print("r2=",r2);

  /* Result
   mae=0.375
   mse=0.5
   r2=0.9486081370449679
   */
```

© 2000-2025, MetaQuotes Ltd.

# ConfusionMatrix

计算混淆矩阵。该方法应用于预测值向量。

```
matrix vector::ConfusionMatrix(
   const vector&        vect_true       // 真值向量
   );


matrix vector::ConfusionMatrix(
   const vector&        vect_true,      // 真值向量
   uint                 label           //选项卡值
   );
```

参数

*vect_true*

[in]真值向量。

*label*

[in]计算混淆矩阵的标签值。

返回值

混淆矩阵。如果未指定标签值，则返回多类混淆矩阵，其中每个标签单独与其他标签匹配。如果指定了标签值，则返回一个2 x 2矩阵，其中指定的标签被视为正数，而所有其他标签均为负数 (ovr，一vs其余)。

注意

混淆矩阵C使得Cij等于已知属于组i且预测属于组j的观察值的数量。因此，在二元分类中，真阴性(TN)的计数为C00，假阴性(FN)为C10，真阳性(TP)为C11，假阳性(FP)为C01。

换句话说，矩阵可以用图形表示如下：

| TN | FP |
| --- | --- |
| FN | TP |

真实值向量和预测值向量的大小应该相同。

示例：

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};
   matrix confusion=y_pred.ConfusionMatrix(y_true);
   Print(confusion);
   confusion=y_pred.ConfusionMatrix(y_true,0);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print(confusion);
   confusion=y_pred.ConfusionMatrix(y_true,1);
   Print(confusion);
   confusion=y_pred.ConfusionMatrix(y_true,2);
   Print(confusion);


/*
  [[3,0,0,0,0,0,0,0,0,0]
   [0,3,0,0,0,0,0,0,0,0]
   [0,0,1,0,1,0,0,1,0,0]
   [0,0,0,1,0,0,0,1,0,0]
   [0,0,1,0,3,0,0,0,0,1]
   [0,0,0,0,0,2,0,0,0,0]
   [1,0,0,0,0,1,1,0,0,0]
   [0,0,0,0,0,0,0,2,0,1]
   [0,0,1,0,0,0,0,0,0,1]
   [0,0,0,0,0,0,0,0,0,4]]
  [[26,1]
   [0,3]]
  [[27,0]
   [0,3]]
  [[25,2]
   [2,1]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# ConfusionMatrixMultilabel

计算每个标签的混淆矩阵。该方法应用于预测值向量。

```
uint vector::ConfusionMatrixMultiLabel(
   const vector&        vect_true,     // 真值向量
   matrix&              confusions[]   // 计算混淆矩阵的数组
   );
```

参数

*vect_true*

[in] 真值向量。

*confusions*

[out] 一个2 ×2矩阵数组，为每个标签计算混淆矩阵。

返回值

计算的混淆矩阵数组的大小。如果失败，则返回0

注意

结果数组可以是动态也可以是静态。如果是静态数组，那么其大小必须不小于类的数量。

真实值向量和预测值向量的大小应该相同。

示例：

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};
   matrix label_confusions[12];

   uint   res=y_pred.ConfusionMatrixMultiLabel(y_true,label_confusions);
   Print("res=",res,"  size=",label_confusions.Size());
   for(uint i=0; i<res; i++)
      Print(label_confusions[i]);


/*
  res=10  size=12
  [[26,1]
   [0,3]]
  [[27,0]
   [0,3]]
  [[25,2]
   [2,1]]
  [[28,0]
```

© 2000-2025, MetaQuotes Ltd.

```
    [1,1]]
   [[24,1]
    [2,3]]
   [[27,1]
    [0,2]]
   [[27,0]
    [2,1]]
   [[25,2]
    [1,2]]
   [[28,0]
    [2,0]]
   [[23,3]
    [0,4]]
*/
```

© 2000-2025, MetaQuotes Ltd.

# ClassificationMetric

计算分类指标来评估预测数据相对于真实数据的质量。该方法适用于预测值的向量。

```
vector vector::ClassificationMetric(
   const vector&                vect_true,      // 真值向量
   ENUM_CLASSIFICATION_METRIC metric          // 指标类型
   );


vector vector::ClassificationMetric(
   const vector&                vect_true,      // 真值向量
   ENUM_CLASSIFICATION_METRIC metric          // 指标类型
   ENUM_AVERAGE_MODE          mode            // 平均模式
   );
```

参数

*vect_true*

[in] 真值向量。

*metric*

[in] ENUM_CLASSIFICATION_METRIC枚举的指标类型。应用CLASSIFICATION_TOP_K_ACCURACY，CLASSIFICATION_AVERAGE_PRECISION和CLASSIFICATION_ROC_AUC（在ClassificationScore方法中使用）以外的值。

*mode*

[in] ENUM_AVERAGE_MODE枚举的平均模式。用于CLASSIFICATION_F1，CLASSIFICATION_JACCARD，CLASSIFICATION_PRECISION和CLASSIFICATION_RECALL指标。

返回值

包含计算指标的向量。在AVERAGE_NONE平均模式的情况下，向量包含每个类别的指标值，而不进行平均。（例如，在二元分类的情况下，这将是分别用于'false'和'true'的两个指标）。

关于平均模式的说明

AVERAGE_BINARY只对二元分类有意义。

AVERAGE_MICRO —通过计算真阳性、假阴性和假阳性总数来计算全局指标。

AVERAGE_MACRO —计算每个标签的指标并找到其未加权平均值。这没有考虑标签不平衡。

AVERAGE_WEIGHTED — 计算每个标签的指标并找到其按支持度加权的平均值（每个标签的真实实例数）。这改变了 'macro'以解决标签不平衡的问题；它可能会导致不在精确率和召回率之间的F分数。

示例：

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
```

© 2000-2025, MetaQuotes Ltd.

```
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};

   vector accuracy=y_pred.ClassificationMetric(y_true,CLASSIFICATION_ACCURACY);
   Print("accuracy=",accuracy);
   vector balanced=y_pred.ClassificationMetric(y_true,CLASSIFICATION_BALANCED_ACCURACY
   Print("balanced=",balanced);
   Print("");

   vector f1_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_MICRO)
   Print("f1_micro=",f1_micro);
   vector f1_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_MACRO)
   Print("f1_macro=",f1_macro);
   vector f1_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_WEI
   Print("f1_weighted=",f1_weighted);
   vector f1_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_NONE);
   Print("f1_none=",f1_none);
   Print("");

   vector jaccard_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVER
   Print("jaccard_micro=",jaccard_micro);
   vector jaccard_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVER
   Print("jaccard_macro=",jaccard_macro);
   vector jaccard_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,A
   Print("jaccard_weighted=",jaccard_weighted);
   vector jaccard_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVERA
   Print("jaccard_none=",jaccard_none);
   Print("");

   vector precision_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION,
   Print("precision_micro=",precision_micro);
   vector precision_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION,
   Print("precision_macro=",precision_macro);
   vector precision_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISI
   Print("precision_weighted=",precision_weighted);
   vector precision_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION,A
   Print("precision_none=",precision_none);
   Print("");

   vector recall_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERAG
   Print("recall_micro=",recall_micro);
   vector recall_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERAG
   Print("recall_macro=",recall_macro);
   vector recall_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVE
   Print("recall_weighted=",recall_weighted);
   vector recall_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERAGE
   Print("recall_none=",recall_none);
   Print("");

//--- 二元分类
```

© 2000-2025, MetaQuotes Ltd.

```
   vector y_pred_bin={0,1,0,1,1,0,0,0,1};
   vector y_true_bin={1,0,0,0,1,0,1,1,1};

   vector f1_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_F1,AVERAGE_
   Print("f1_bin=",f1_bin);
   vector jaccard_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_JACCAI
   Print("jaccard_bin=",jaccard_bin);
   vector precision_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_PREC
   Print("precision_bin=",precision_bin);
   vector recall_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_RECALL,
   Print("recall_bin=",recall_bin);


/*
  accuracy=[0.6666666666666666]
  balanced=[0.6433333333333333]

  f1_micro=[0.6666666666666666]
  f1_macro=[0.6122510822510823]
  f1_weighted=[0.632049062049062]
  f1_none=[0.8571428571428571,1,0.3333333333333333,0.6666666666666666,0.6666666666666

  jaccard_micro=[0.5]
  jaccard_macro=[0.4921428571428572]
  jaccard_weighted=[0.5056349206349205]
  jaccard_none=[0.75,1,0.2,0.5,0.5,0.6666666666666666,0.3333333333333333,0.4,0,0.5714

  precision_micro=[0.6666666666666666]
  precision_macro=[0.6571428571428571]
  precision_weighted=[0.6706349206349207]
  precision_none=[0.75,1,0.3333333333333333,1,0.75,0.6666666666666666,1,0.5,0,0.571428

  recall_micro=[0.6666666666666666]
  recall_macro=[0.6433333333333333]
  recall_weighted=[0.6666666666666666]
  recall_none=[1,1,0.3333333333333333,0.5,0.6,1,0.3333333333333333,0.6666666666666666,

  f1_bin=[0.4444444444444445]
  jaccard_bin=[0.2857142857142857]
  precision_bin=[0.5]
  recall_bin=[0.4]
*/
```

© 2000-2025, MetaQuotes Ltd.

# ClassificationScore

计算分类指标来评估预测数据相对于真实数据的质量。

与机器学习部分中的其他方法不同，这个指标适用于真实值的向量，而不是预测值的向量。

```
vector vector::ClassificationScore(
   const matrix&                pred_scores,    // 包含每一类概率分布的矩阵
   ENUM_CLASSIFICATION_METRIC   metric          // 指标类型
   ENUM_AVERAGE_MODE            mode            // 平均模式
   );


vector vector::ClassificationScore(
   const matrix&                pred_scores,    // 包含每一类概率分布的矩阵
   ENUM_CLASSIFICATION_METRIC   metric          // 指标类型
   int                          param           // 附加参数
   );
```

参数

*pred_scores*

[in] 包含一组水平向量和每一类概率的矩阵。矩阵行数应与真值向量的大小相对应。

*metric*

[in] ENUM_CLASSIFICATION_METRIC枚举的指标类型。使用CLASSIFICATION_TOP_K_ACCURACY，CLASSIFICATION_AVERAGE_PRECISION和CLASSIFICATION_ROC_AUC值。

*mode*

[in] ENUM_AVERAGE_MODE枚举的平均模式。用于CLASSIFICATION_AVERAGE_PRECISION和CLASSIFICATION_ROC_AUC指标。

*param*

[in] 对于CLASSIFICATION_TOP_K_ACCURACY指标，应指定整数K值来替代平均模式。

返回值

包含计算指标的向量。在AVERAGE_NONE平均模式的情况下，向量包含每个类别的指标值，而不进行平均。例如，在二元分类的情况下，这将是分别用于'false'和'true'的两个指标）。

关于平均模式的说明

AVERAGE_BINARY只对二元分类有意义。

AVERAGE_MICRO —通过将标签指标矩阵的每个元素视为一个标签来计算全局指标。标签指标矩阵是指一个矩阵，其中包含每个标签的一组概率。

AVERAGE_MACRO —计算每个标签的指标并找到其未加权平均值。这没有考虑标签不平衡。

© 2000-2025, MetaQuotes Ltd.

AVERAGE_WEIGHTED — 计算每个标签的指标并找到其按支持度加权的平均值（每个标签的真实实例数）。

注意

在二元分类的情况下，我们不仅可以输入一个n x 2矩阵，其中第一列包含负标签的概率，第二列包含正标签的概率，还可以输入一个由一列正概率组成的矩阵。这是因为二元分类模型可以返回正标签的两个概率或一个概率。

示例：

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   //vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};

//--- label scores          0         1         2         3         4         5
   matrix y_scores={{0.000109, 0.000186, 0.000449, 0.000052, 0.000002, 0.000022, 0.000
                    {0.000091, 0.081956, 0.916816, 0.001106, 0.000006, 0.000002, 0.000
                    {0.000108, 0.972863, 0.003600, 0.000021, 0.010479, 0.000015, 0.000
                    {0.925425, 0.000080, 0.002913, 0.000057, 0.000274, 0.000638, 0.061
                    {0.000060, 0.000126, 0.000006, 0.000000, 0.993513, 0.000000, 0.000
                    {0.000016, 0.982124, 0.000045, 0.000002, 0.008445, 0.000001, 0.000
                    {0.000000, 0.000040, 0.000001, 0.000000, 0.989395, 0.000167, 0.000
                    {0.000795, 0.002938, 0.023447, 0.007418, 0.021838, 0.002476, 0.000
                    {0.000091, 0.000226, 0.000038, 0.000007, 0.000048, 0.854910, 0.068
                    {0.000000, 0.000000, 0.000000, 0.000000, 0.003004, 0.000000, 0.000
                    {0.998856, 0.000009, 0.000976, 0.000002, 0.000000, 0.000013, 0.000
                    {0.000178, 0.000446, 0.000326, 0.000033, 0.000193, 0.000071, 0.998
                    {0.000005, 0.000016, 0.000153, 0.000045, 0.004110, 0.000012, 0.000
                    {0.994188, 0.000003, 0.002584, 0.000005, 0.000005, 0.000100, 0.000
                    {0.000173, 0.990569, 0.000792, 0.000040, 0.001798, 0.000035, 0.000
                    {0.000000, 0.000537, 0.000008, 0.005080, 0.000046, 0.992910, 0.000
                    {0.000127, 0.000003, 0.000003, 0.000000, 0.001583, 0.000000, 0.000
                    {0.000001, 0.000012, 0.000072, 0.000020, 0.000000, 0.000000, 0.000
                    {0.000020, 0.000105, 0.001139, 0.901343, 0.002132, 0.083873, 0.000
                    {0.000002, 0.000048, 0.000019, 0.000000, 0.999347, 0.000002, 0.000
                    {0.000059, 0.001344, 0.612502, 0.002749, 0.000229, 0.000678, 0.000
                    {0.000586, 0.000740, 0.001625, 0.000007, 0.269341, 0.000076, 0.016
                    {0.009547, 0.018055, 0.283795, 0.071079, 0.426074, 0.082335, 0.036
                    {0.002506, 0.002545, 0.001148, 0.005659, 0.020416, 0.000112, 0.006
                    {0.001263, 0.001769, 0.000293, 0.000011, 0.000302, 0.881768, 0.112
                    {0.002904, 0.002909, 0.013421, 0.001461, 0.007519, 0.001251, 0.000
                    {0.000055, 0.001080, 0.893158, 0.000000, 0.104492, 0.000159, 0.001
                    {0.000344, 0.002693, 0.071184, 0.000262, 0.000001, 0.000003, 0.000
                    {0.001404, 0.009375, 0.002638, 0.229189, 0.000064, 0.000896, 0.007
                    {0.491140, 0.000125, 0.000024, 0.000302, 0.000038, 0.034947, 0.473

   vector top_k=y_true.ClassificationScore(y_scores,CLASSIFICATION_TOP_K_ACCURACY,1);
   Print("top 1 accuracy score = ",top_k);
   top_k=y_true.ClassificationScore(y_scores,CLASSIFICATION_TOP_K_ACCURACY,2);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("top 2 accuracy score = ",top_k);
   vector y_true2={0, 1, 2, 2};
   matrix y_score2={{0.5, 0.2, 0.2},  // 0 is in top 2
                    {0.3, 0.4, 0.2},  // 1 is in top 2
                    {0.2, 0.4, 0.3},  // 2 is in top 2
                    {0.7, 0.2, 0.1}}; // 2 isn't in top 2
   top_k=y_true2.ClassificationScore(y_score2,CLASSIFICATION_TOP_K_ACCURACY,2);
   Print("top k = ",top_k);
   Print("");

   vector ap_micro=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISIO
   Print("average precision score micro = ",ap_micro);
   vector ap_macro=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISIO
   Print("average precision score macro = ",ap_macro);
   vector ap_weighted=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECI
   Print("average precision score weighted = ",ap_weighted);
   vector ap_none=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISION
   Print("average precision score none = ",ap_none);
   Print("");

   vector area_micro=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERAG
   Print("roc auc score micro = ",area_micro);
   vector area_macro=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERAG
   Print("roc auc score macro = ",area_macro);
   vector area_weighted=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVE
   Print("roc auc score weighted = ",area_weighted);
   vector area_none=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERAGE
   Print("roc auc score none = ",area_none);
   Print("");

//--- 二元分类
   vector y_pred_bin={0,1,0,1,1,0,0,0,1};
   vector y_true_bin={1,0,0,0,1,0,1,1,1};
   vector y_score_true={0.3,0.7,0.1,0.6,0.9,0.0,0.4,0.2,0.8};
   matrix y_score1_bin(y_score_true.Size(),1);
   y_score1_bin.Col(y_score_true,0);
   matrix y_scores_bin={{0.7, 0.3},
                        {0.3, 0.7},
                        {0.9, 0.1},
                        {0.4, 0.6},
                        {0.1, 0.9},
                        {1.0, 0.0},
                        {0.6, 0.4},
                        {0.8, 0.2},
                        {0.2, 0.8}};

   vector ap=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_AVERAGE_PRECIS
   Print("average precision score binary = ",ap);
   vector ap2=y_true_bin.ClassificationScore(y_score1_bin,CLASSIFICATION_AVERAGE_PRECI
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("average precision score binary = ",ap2);
   vector ap3=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_AVERAGE_PREC
   Print("average precision score none = ",ap3);
   Print("");

   vector area=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_ROC_AUC,AVE
   Print("roc auc score binary = ",area);
   vector area2=y_true_bin.ClassificationScore(y_score1_bin,CLASSIFICATION_ROC_AUC,AV
   Print("roc auc score binary = ",area2);
   vector area3=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_ROC_AUC,AV
   Print("roc auc score none = ",area3);


/*
  top 1 accuracy score = [0.6666666666666666]
  top 2 accuracy score = [1]
  top k = [0.75]

  average precision score micro = [0.8513333333333333]
  average precision score macro = [0.9326666666666666]
  average precision score weighted = [0.9333333333333333]
  average precision score none = [1,1,0.7,1,0.9266666666666666,0.8333333333333333,1,0.

  roc auc score micro = [0.9839506172839506]
  roc auc score macro = [0.9892068783068803]
  roc auc score weighted = [0.9887354497354497]
  roc auc score none = [1,1,0.9506172839506173,1,0.984,0.9821428571428571,1,0.9753086

  average precision score binary = [0.7961904761904761]
  average precision score binary = [0.7961904761904761]
  average precision score none = [0.7678571428571428,0.7961904761904761]

  roc auc score binary = [0.7]
  roc auc score binary = [0.7]
  roc auc score none = [0.7,0.7]
*/
```

© 2000-2025, MetaQuotes Ltd.

# PrecisionRecall

计算值以构建精确调用曲线。与ClassificationScore类似，此方法应用于真值向量。

```
bool vector::PrecisionRecall(
   const matrix&                 pred_scores,    // 包含每类概率分布的矩阵
   const ENUM_ENUM_AVERAGE_MODE  mode            // 平均模式
   matrix&                       precision,      // 计算每个阈值的精确值
   matrix&                       recall,         // 计算每个阈值的调用值
   matrix&                       thresholds,     // 以降序排列阈值
   );
```

参数

*pred_scores*

[in] 包含一组水平向量（含有每个类的概率）的矩阵。矩阵行数必须与真值向量的大小相对应。

*模式*

[in] ENUM_AVERAGE_MODE枚举的平均模式。只使用AVERAGE_NONE，AVERAGE_BINARY和AVERAGE_MICRO。

*precision*

[out] 包含计算精度曲线值的矩阵。如果不使用平均模式（AVERAGE_NONE），则矩阵中的行数对应于模型类的数量。列数对应于真值向量的大小（或概率分布矩阵pred_score中的行数）。在微平均的情况下，矩阵中的行数对应于阈值的总数，不包括重复项。

*recall*

[out] 包含计算调用曲线值的矩阵。

*threshold*

[out] 对概率矩阵进行排序得到的阈值矩阵

注意

请参阅ClassificationScore方法的注释。

例如

从mnist.onnx模型收集统计数据的示例（99%准确率）。

```
//--- 分类指标数据
   vectorf y_true(images);
   vectorf y_pred(images);
   matrixf y_scores(images,10);
//--- 输入输出
   matrixf image(28,28);
   vectorf result(10);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 测试
   for(int test=0; test<images; test++)
     {
      image=test_data[test].image;
      if(!OnnxRun(model,ONNX_DEFAULT,image,result))
        {
         Print("OnnxRun error ",GetLastError());
         break;
        }
      result.Activation(result,AF_SOFTMAX);
     //--- 收集数据
      y_true[test]=(float)test_data[test].label;
      y_pred[test]=(float)result.ArgMax();
      y_scores.Row(result,test);
     }    }
```

## Accuracy calculation

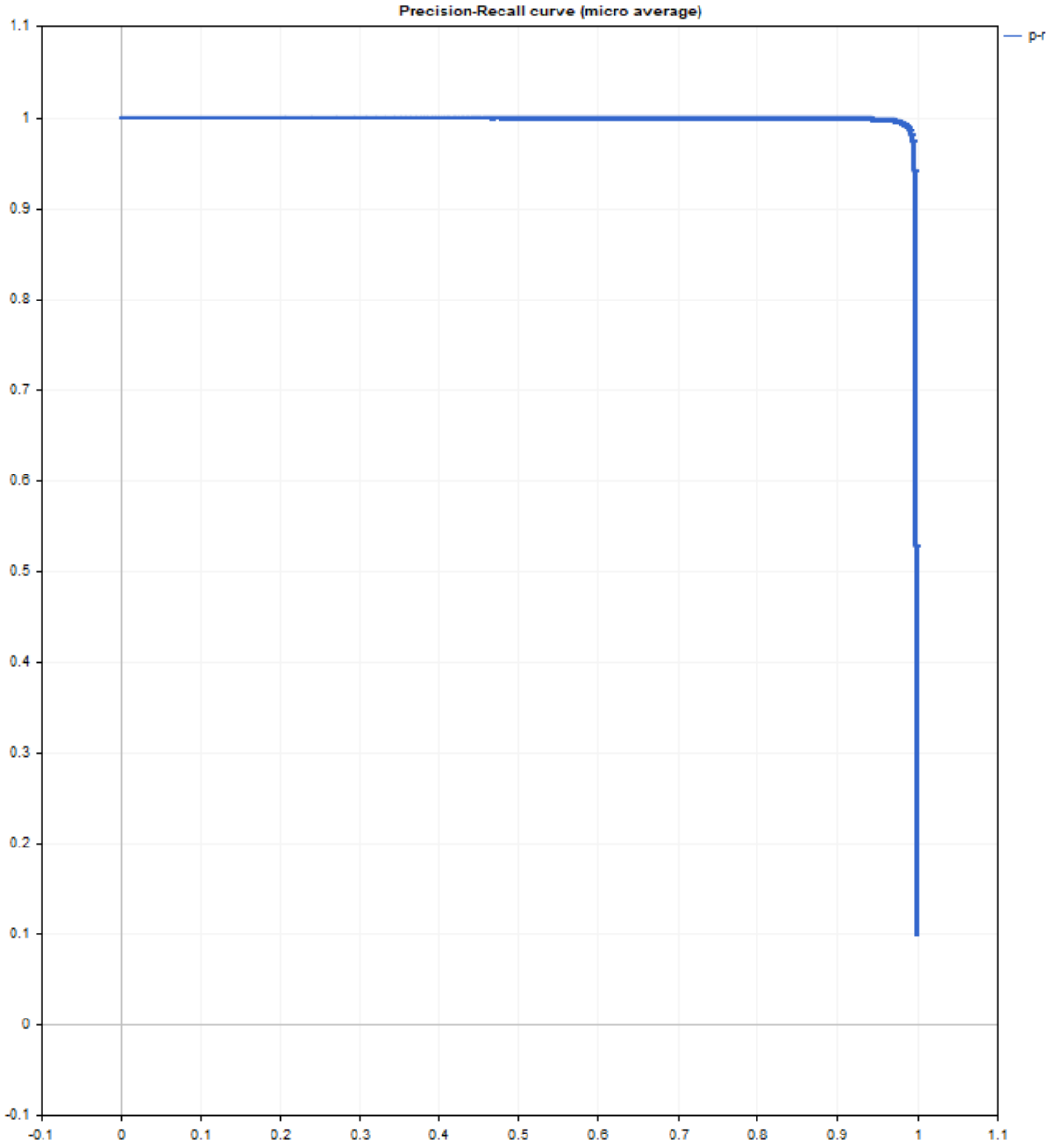
```
   vectorf accuracy=y_pred.ClassificationMetric(y_true,CLASSIFICATION_ACCURACY);
   PrintFormat("accuracy=%f",accuracy[0]);

accuracy=0.989000
```

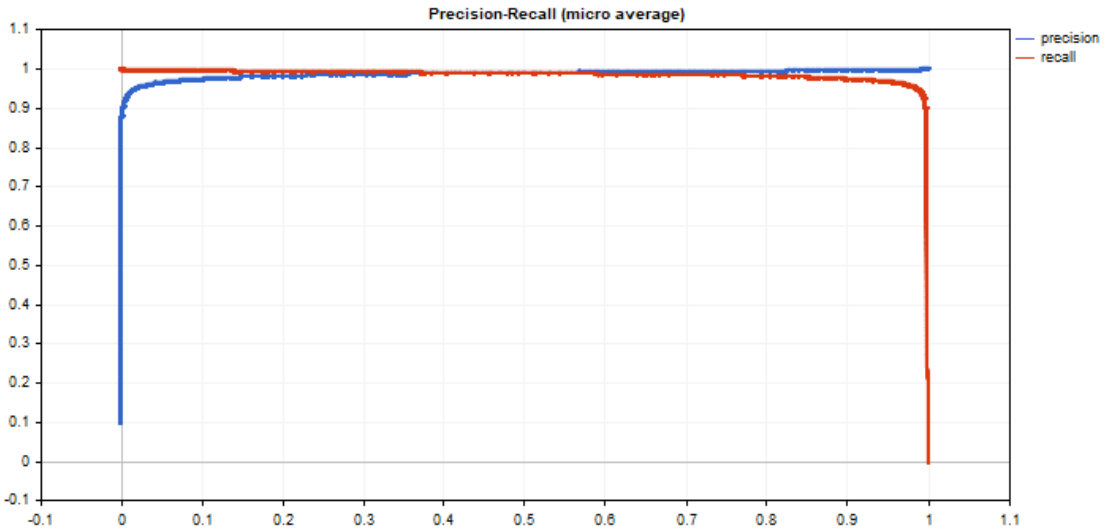
绘制精确调用图的示例，其中精确值绘制在y轴上，调用值绘制在x轴上。此外，精度图和调用图也分别绘制，阈值绘制在x轴上

```
   if(y_true.PrecisionRecall(y_scores,AVERAGE_MICRO,mat_precision,mat_recall,mat_thres
     {
      double precision[],recall[],thres[];
      ArrayResize(precision,mat_thres.Cols());
      ArrayResize(recall,mat_thres.Cols());
      ArrayResize(thres,mat_thres.Cols());

      for(uint i=0; i<thres.Size(); i++)
        {
         precision[i]=mat_precision[0][i];
         recall[i]=mat_recall[0][i];
         thres[i]=mat_thres[0][i];
        }
      thres[0]=thres[1]+0.001;

      PlotCurve("Precision-Recall curve (micro average)","p-r","",recall,precision);
      Plot2Curves("Precision-Recall (micro average)","precision","recall",thres,precis
     }
```

**Resulting curves:**

© 2000-2025, MetaQuotes Ltd.




© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# ReceiverOperatingCharacteristic

计算值以构建接收器工作特性(ROC)曲线。与ClassificationScore类似，此方法应用于真值向量。

```
bool vector::ReceiverOperatingCharacteristic(
   const matrix&                 pred_scores,    // 包含每类概率分布的矩阵
   const ENUM_ENUM_AVERAGE_MODE  mode            // 平均模式
   matrix&                       fpr,            // 计算每个阈值的假阳性率值
   matrix&                       tpr,            // 计算每个阈值的真阳性率值
   matrix&                       thresholds,     // 以降序排列阈值
   );
```

参数

*pred_scores*

[in] 包含一组水平向量（含有每个类的概率）的矩阵。矩阵行数必须与真值向量的大小相对应。

*模式*

[in] ENUM_AVERAGE_MODE枚举的平均模式。只使用AVERAGE_NONE，AVERAGE_BINARY和AVERAGE_MICRO。

*fpr*

[out] 假阳性率曲线计算值的矩阵。如果不使用平均模式（AVERAGE_NONE），则矩阵中的行数对应于模型类的数量。列数对应于真值向量的大小（或概率分布矩阵pred_score中的行数）。在微平均的情况下，矩阵中的行数对应于阈值的总数，不包括重复项。

*tpr*

[out] 真阳性率曲线计算值的矩阵。

*threshold*

[out] 对概率矩阵进行排序得到的阈值矩阵

注意

请参阅ClassificationScore方法的注释。

例如

绘制ROC图的示例，其中tpr值绘制在y轴上，fpr值绘制在x轴上。此外，fpr图和tpr图也分别绘制，阈值绘制在x轴上

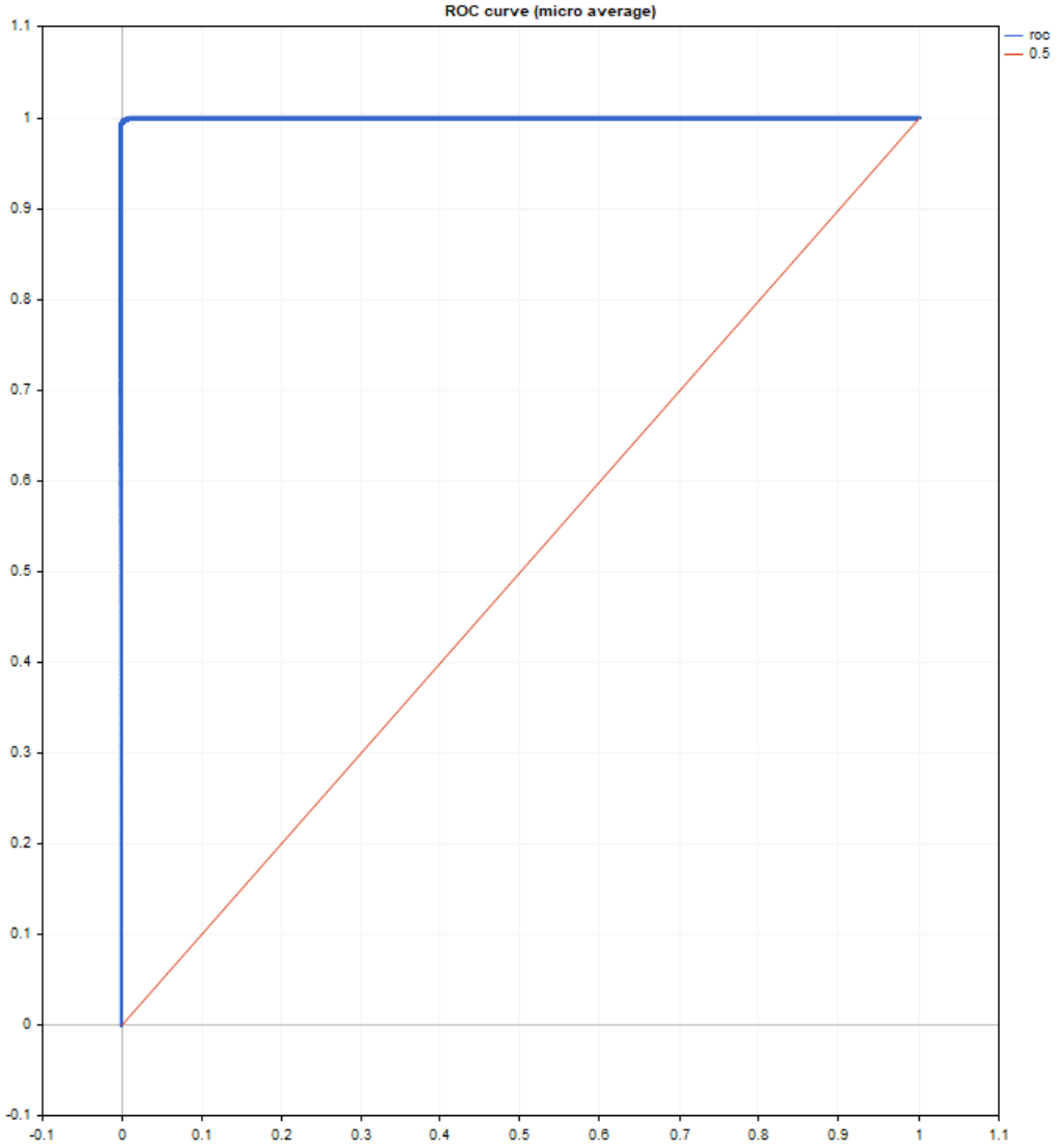
```
   matrixf mat_thres;
   matrixf mat_fpr;
   matrixf mat_tpr;

   if(y_true.ReceiverOperatingCharacteristic(y_scores,AVERAGE_MICRO,mat_fpr,mat_tpr,ma
     {
      double fpr[],tpr[],thres[];
```

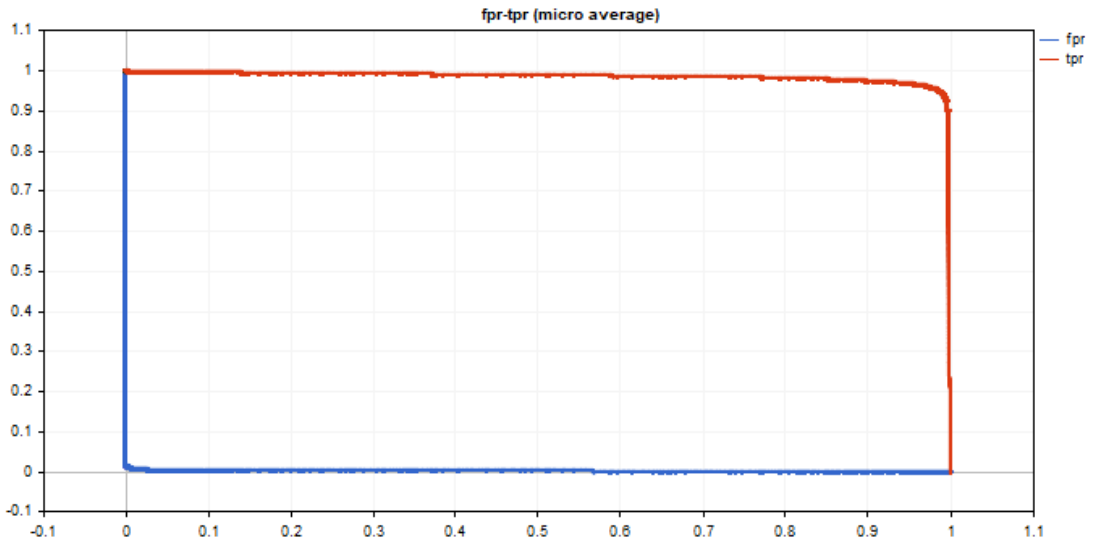
© 2000-2025, MetaQuotes Ltd.

```
   ArrayResize(fpr,mat_thres.Cols());
   ArrayResize(tpr,mat_thres.Cols());
   ArrayResize(thres,mat_thres.Cols());

   for(uint i=0; i<fpr.Size(); i++)
     {
      fpr[i]=mat_fpr[0][i];
      tpr[i]=mat_tpr[0][i];
      thres[i]=mat_thres[0][i];
     }
   thres[0]=thres[1]+0.001;

   PlotCurve("ROC curve (micro average)","roc","0.5",fpr,tpr);
   Plot2Curves("fpr-tpr (micro average)","fpr","tpr",thres,fpr,tpr);
  }
```

结果曲线：

© 2000-2025, MetaQuotes Ltd.




© 2000-2025, MetaQuotes Ltd.

图形输出代码很简单，并且基于<Graphics/Graphic.mqh>标准库。

示例使用mnist.onnx模型的数据。代码在PrecisionRecall方法描述中展示。

ROC AUC接近理想值。

```
roc auc score micro = [0.99991]
```

© 2000-2025, MetaQuotes Ltd.

# OpenBLAS Methods

OpenBLAS is a high-performance open-source linear algebra library that implements BLAS (Basic Linear Algebra Subprograms) and some LAPACK functions. OpenBLAS is designed to improve computational performance, particularly in matrix and vector operations, which are often used in scientific and engineering tasks such as machine learning, numerical methods, and simulations.

Key features of OpenBLAS:

- Multithreading support: OpenBLAS can efficiently use multiple processor cores for parallel computations, significantly accelerating operations on multiprocessor systems.
- Optimization for processor architectures: OpenBLAS includes optimized builds for various processors such as Intel, AMD, ARM and others. The library automatically detects processor characteristics and selects the most suitable function implementations.
- Extensive BLAS operation support: OpenBLAS implements core BLAS functions, including vector operations (e.g., vector addition and dot product), matrix operations (multiplication), and vector-matrix operations.
- LAPACK compatibility: The library supports LAPACK (Linear Algebra PACKage) functions for more complex linear algebra operations, such as solving systems of linear equations, calculating matrix eigenvalues, and others.
- High performance: Compared to other BLAS libraries, OpenBLAS often shows better results due to hand-optimization for specific processor architectures.

## Applications

OpenBLAS is widely used in applications involving numerical computations:

- Training neural networks and other machine learning tasks.
- Scientific computing (e.g. modeling of physical processes).
- Processing and analyzing large amounts of data.

The library is integrated into many popular scientific software packages such as NumPy, SciPy, and TensorFlow, which rely on high-performance linear algebra operations.

OpenBLAS is an excellent choice for those seeking an open-source solution for high-performance computing, particularly when working with large matrices and vectors.

| Function | Action |
| --- | --- |
| SingularValueDecompositionDC | Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD). |
| SingularValueDecompositionQR | Singular Value Decomposition, QR algorithm. This algorithm is considered a classical SVD algorithm (lapack function GESVD). |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| SingularValueDecompositionQRPivot | Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ). |
| SingularValueDecompositionBisect | Singular Value Decomposition, bisection algorithm (lapack function GESVDX). |
| SingularValueDecompositionJacobiHigh | Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV). |
| SingularValueDecompositionJacobiLow | Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases. |
| SingularValueDecompositionBidiagDC | Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSVDX). |
| SingularValueDecompositionBidiagBisect | Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX). |
| EigenSolver | Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV). |
| EigenSolver2 | Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV). |
| EigenSolverX | Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX). |
| EigenSolverShur | Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition. |
| EigenSymmetricDC | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD). |
| EigenSymmetricQR | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| | conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV). |
| EigenSymmetricRobust | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR). |
| EigenSymmetricBisect | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX). |
| SingularSpectrumAnalysisSpectrum | A method function for calculating the relative contributions of spectral components based on their eigenvalues. |
| SingularSpectrumAnalysisForecast | A method function for calculating reconstructed and predicted data using spectral components of the input time series. |
| SingularSpectrumAnalysisReconstructComponents | A method function for calculating reconstructed components of the input time series and their contributions. |
| SingularSpectrumAnalysisReconstructSeries | A method function for calculating the reconstructed time series using the first component_count components. |

© 2000-2025, MetaQuotes Ltd.

# Singular value decomposition

This section features functions for decomposing a matrix into three components: orthogonal matrices and a diagonal matrix of singular values. SVD is applied to solve various linear algebra problems such as data dimensionality reduction, image compression, solving systems of equations, and data analysis and optimization. The main functions allow you to compute singular values and vectors, reconstruct matrices, and approximate matrices with reduced rank accuracy.

| Function | Action |
| --- | --- |
| SingularValueDecompositionDC | Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD). |
| SingularValueDecompositionQR | Singular Value Decomposition, QR algorithm. This algorithm is considered a classical SVD algorithm (lapack function GESVD). |
| SingularValueDecompositionQRPivot | Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ). |
| SingularValueDecompositionBisect | Singular Value Decomposition, bisection algorithm (lapack function GESVDX). |
| SingularValueDecompositionJacobiHigh | Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV). |
| SingularValueDecompositionJacobiLow | Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases. |
| SingularValueDecompositionBidiagDC | Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSVDX). |
| SingularValueDecompositionBidiagBisect | Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX). |
| SingularValueDecompositionBidiagQR | Computes the singular value decomposition of a general matrix that has been reduced to bidiagonal form by the method ReduceToBidiagonal. Lapack function BDSQR. |

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionDC

Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z      jobz,     // how to computed
   vector&         S,        // vector of computed singular values
   matrix&         U,        // matrix of computed left vectors U
   matrix&         VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z      jobz,     // how to computed
   vectorf&        S,        // vector of computed singular values
   matrixf&        U,        // matrix of computed left vectors U
   matrixf&        VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z      jobz,     // how to computed
   vector&         S,        // vector of computed singular values
   matrixc&        U,        // matrix of computed left vectors U
   matrixc&        VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z      jobz,                  // how to computed
   vectorf&        singular_values,       // vector of computed singular values
   matrixcf&       U,                     // matrix of computed left vectors U
   matrixcf&       VT                     // transposed matrix of right vectors VT
   );
```

### Parameters

*jobz*

[in] ENUM_SVD_Z enumeration value which determines the method for computing left and singular eigenvectors.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

© 2000-2025, MetaQuotes Ltd.

*VT*

[out] Matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobz parameter.

When jobv is set to SVDZ_N, the left and right vectors are not computed. Only singular values are computed.

When jobv is set to SVDZ_A, the full matrices of the U and VT vectors are computed.

When the value is SVDZ_S, truncated matrices of vectors U and VT are computed.

**ENUM_SVD_Z**

An enumeration defining the way to compute left and right singular vectors.

| ID | Description |
|---|---|
| SVDZ_N | Columns U or rows VT are not computed |
| SVDZ_A | All M columns of U or all N columns of VT are returned in arrays U and VT |
| SVDZ_S | The first min(M,N) columns of U or the first min(M,N) columns of VT are returned in arrays U and VT |

**See also**

SingularValueDecompositionQR, SingularValueDecompositionQRPivot

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionQR

Singular Value Decomposition, QR algorithm. Considered a classical SVD algorithm (lapack function GESVD).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z      jobu,     // how to compute left vectors
   ENUM_SVD_Z      jobv,     // how to compute right vectors
   vector&         S,        // vector of computed singular values
   matrix&         U,        // matrix of computed left vectors U
   matrix&         VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z      jobu,     // how to compute left vectors
   ENUM_SVD_Z      jobv,     // how to compute right vectors
   vectorf&        S,        // vector of computed singular values
   matrixf&        U,        // matrix of computed left vectors U
   matrixf&        VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z      jobu,     // how to compute left vectors
   ENUM_SVD_Z      jobv,     // how to compute right vectors
   vector          S,        // vector of computed singular values
   matrixc         U,        // matrix of computed left vectors U
   matrixc         VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z      jobu,                  // how to compute left vectors
   ENUM_SVD_Z      jobv,                  // how to compute right vectors
   vectorf&        singular_values,       // vector of computed singular values
   matrixcf&       u,                     // matrix of computed left vectors U
   matrixcf&       vt                     // transposed matrix of right vectors VT
   );
```

### Parameters

*jobu*

[in] ENUM_SVD_Z enumeration value defining how the left singular vectors should be computed.

*jobv*

© 2000-2025, MetaQuotes Ltd.

[in] ENUM_SVD_Z enumeration value defining how the right singular vectors should be computed.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

*VT*

[out] Matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobu and jobv parameters.

When set to SVDZ_N, the left (jobu) and right (jobv) vectors are not computed. Singular values are always computed.

When set to SVDZ_A, the full matrices of vectors U (jobu) or VT (jobv) are computed.

With the value SVDZ_S, truncated matrices of vectors U (jobu) or VT (jobv) are computed.

**ENUM_SVD_Z**

An enumeration defining the way to compute left and right singular vectors.

| ID | Description |
|---|---|
| SVDZ_N | Columns U or rows VT are not computed |
| SVDZ_A | All M columns of U or all N columns of VT are returned in arrays U and VT |
| SVDZ_S | The first min(M,N) columns of U or the first min(M,N) columns of VT are returned in arrays U and VT |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQRPivot

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionQRPivot

Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A   joba,     // computation accuracy level
   ENUM_SVDQRP_P   jobp,     // use row reversal to compute
   ENUM_SVDQRP_R   jobr,     // use triangular matrix R to compute
   ENUM_SVDQRP_U   jobu,     // how to compute left vectors
   ENUM_SVDQRP_V   jobv,     // how to compute right vectors
   vector&         S,        // vector of computed singular values
   matrix&         U,        // matrix of computed left vectors U
   matrix&         VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A   joba,     // computation accuracy level
   ENUM_SVDQRP_P   jobp,     // use row reversal to compute
   ENUM_SVDQRP_R   jobr,     // use triangular matrix R to compute
   ENUM_SVDQRP_U   jobu,     // how to compute left vectors
   ENUM_SVDQRP_V   jobv,     // how to compute right vectors
   vectorf&        S,        // vector of computed singular values
   matrixf&        U,        // matrix of computed left vectors U
   matrixf&        VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A   joba,     // computation accuracy level
   ENUM_SVDQRP_P   jobp,     // use row reversal to compute
   ENUM_SVDQRP_R   jobr,     // use triangular matrix R to compute
   ENUM_SVDQRP_U   jobu,     // how to compute left vectors
   ENUM_SVDQRP_V   jobv,     // how to compute right vectors
   vector&         S,        // vector of computed singular values
   matrixc&        U,        // matrix of computed left vectors U
   matrixc&        VT        // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A   joba,                  // computation accuracy level
   ENUM_SVDQRP_P   jobp,                  // use row reversal to compute
   ENUM_SVDQRP_R   jobr,                  // use triangular matrix R to compute
   ENUM_SVDQRP_U   jobu,                  // how to compute left vectors
```

© 2000-2025, MetaQuotes Ltd.

```
   ENUM_SVDQRP_V   jobv,                  // how to compute right vectors
   vectorf&        singular_values,       // vector of computed singular values
   matrixcf&       u,                     // matrix of computed left vectors U
   matrixcf&       vt                     // transposed matrix of right vectors VT
   );
```

**Parameters**

*joba*

[in] ENUM_SVDQRP_A enumeration value defining the accuracy level of the SVD computation.

*jobp*

[in] ENUM_SVDQRP_P enumeration value defining the use of row reversal during the computation process.

*jobr*

[in] ENUM_SVDQRP_R enumeration value defining whether to transpose the triangular matrix R obtained as a result of the initial QR factorization.

*jobu*

[in] ENUM_SVDQRP_U enumeration value that determines how the left singular vectors should be computed.

*jobv*

[in] ENUM_SVDQRP_V enumeration value defining how the right singular vectors should be computed.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

*VT*

[out] Matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The number of matrix rows must not be less than the number of columns.

If both left and right singular vectors are computed, jobv must be set to SVDQRPV_R when jobu=SVDQRPU_R.

**ENUM_SVDQRP_A**

An enumeration that specifies the level of accuracy of the SVD computation.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDQRP_A | The requested accuracy corresponds to the inverse error limited by epsilon. This is an aggressive level of truncation. |
| SVDQRPA_M | Similar to SVDQRP_A, but the truncation is softer. This is the average level of truncation. |
| SVDQRPA_H | High accuracy is required. |

**ENUM_SVDQRP_P**

An enumeration that specifies whether to use row reversal during calculation.

| ID | Description |
| --- | --- |
| SVDQRPP_P | The rows of A are ordered in descending order. Recommended for numerical reliability. |
| SVDQRPP_N | No row reversal. |

**ENUM_SVDQRP_R**

An enumeration that specifies whether to transpose the triangular matrix R obtained as a result of the initial QR factorization.

| ID | Description |
| --- | --- |
| SVDQRPR_T | After the initial rotated QR factorization, GESVD is applied to the transpose R**T of the calculated triangular factor R. |
| SVDQRPR_N | The triangular factor R is given as input to GESVD. |

**ENUM_SVDQRP_U**

An enumeration defining how left singular vectors should be computed.

| ID | Description |
| --- | --- |
| SVDQRPU_A | All M left singular vectors are computed. |
| SVDQRPU_S | The min(M,N) left singular vectors are computed. |
| SVDQRPU_R | The numerical rank NUMRANK is determined and only the NUMRANK of the left singular vectors is computed. |
| SVDQRPU_F | N left singular vectors are returned. |
| SVDQRPU_N | Left singular vectors are not computed. |

**ENUM_SVDQRP_V**

An enumeration defining how right singular vectors should be computed.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDQRPV_A | All N right singular vectors are computed. |
| SVDQRPV_R | The numerical rank NUMRANK is defined and only the NUMRANK of right singular vectors are computed. |
| SVDQRPV_N | Right singular vectors are not computed. |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQR

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionBisect

Singular Value Decomposition, bisection algorithm (lapack function GESVDX).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS     jobv,      // computation accuracy level
   ENUM_BLAS_RANGE      range,     // subset of computable singular values
   double               lower,     // lower limit of the subset
   double               upper,     // upper limit of the subset
   vector&              S,         // vector of computed singular values
   matrix&              U,         // matrix of computed left vectors U
   matrix&              VT         // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS     jobv,      // computation accuracy level
   ENUM_BLAS_RANGE      range,     // subset of computable singular values
   double               lower,     // lower limit of the subset
   double               upper,     // upper limit of the subset
   vectorf&             S,         // vector of computed singular values
   matrixf&             U,         // matrix of computed left vectors U
   matrixf&             VT         // ransposed matrix of right vectors VT
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS     jobv,      // computation accuracy level
   ENUM_BLAS_RANGE      range,     // subset of computable singular values
   double               lower,     // lower limit of the subset
   double               upper,     // upper limit of the subset
   vector               S,         // vector of computed singular values
   matrixc              U,         // matrix of computed left vectors U
   matrixc              VT         // transposed matrix of right vectors VT
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS     jobv,               // computation accuracy level
   ENUM_BLAS_RANGE      range,              // subset of computable singular values
   double               lower,              // lower limit of the subset
   double               upper,              // upper limit of the subset
   vectorf&             singular_values,    // vector of computed singular values
   matrixcf&            u,                  // matrix of computed left vectors U
   matrixcf&            vt                  // transposed matrix of right vectors VT
```

© 2000-2025, MetaQuotes Ltd.

```
   );
```

**Parameters**

*jobv*

[in] ENUM_SVD_VECTORS enumeration value which determines the method for computing left and singular eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

*lower*

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

*upper*

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

*VT*

[out] Matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobuv* and *range* parameters.

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

**ENUM_SVD_VECTORS**

An enumeration defining the way to compute left and right singular vectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDVECTORS_N | Only singular values are computed, without vectors. |
| SVDVECTORS_U | Left singular vectors are computed. |
| SVDVECTORS_V | Right singular vectors are computed. |
| SVDVECTORS_UV | Left and right singular vectors are computed. |

**ENUM_BLAS_RANGE**

An enumeration defining a subset of computable singular values and vectors.

| ID | Description |
| --- | --- |
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | Singular or eigenvalues from IL to IU will be found |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQR, SingularValueDecompositionQRPivot

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionJacobiHigh

Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A   joba,            // computation accuracy level
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   ENUM_SVDJH_R   jobr,            // define range of computable singular values
   ENUM_SVDJH_T   jobt,            // define whether to transpose when computing a sq
   ENUM_SVDJH_P   jobp,            // the possibility of structured perturbations to
   vector&        S,               // vector of computed singular values
   matrix&        U,               // matrix of computed left vectors U
   matrix&        V,               // matrix of computed left vectors V
   vector&        work_results     // additional computation results
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A   joba,            // computation accuracy level
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   ENUM_SVDJH_R   jobr,            // define range of computable singular values
   ENUM_SVDJH_T   jobt,            // define whether to transpose when computing a sq
   ENUM_SVDJH_P   jobp,            // the possibility of structured perturbations to
   vectorf&       S,               // vector of computed singular values
   matrixf&       U,               // U matrix of computed left vectors
   matrixf&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A   joba,            // computation accuracy level
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   ENUM_SVDJH_R   jobr,            // define range of computable singular values
   ENUM_SVDJH_T   jobt,            // define whether to transpose when computing a sq
   ENUM_SVDJH_P   jobp,            // the possibility of structured perturbations to
   vectorc&       S,               // vector of computed singular values
   matrixc&       U,               // U matrix of computed left vectors
   matrixc&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

© 2000-2025, MetaQuotes Ltd.

**Computing for type matrix<complexf>**

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A    joba,                  // computation accuracy level
   ENUM_SVDJH_U    jobu,                  // how to compute left vectors
   ENUM_SVDJH_V    jobv,                  // how to compute right vectors
   ENUM_SVDJH_R    jobr,                  // define range of computable singular valu
   ENUM_SVDJH_T    jobt,                  // define whether to transpose when computi
   ENUM_SVDJH_P    jobp,                  // the possibility of structured perturbati
   vectorf&        singular_values,       // vector of computed singular values S
   matrixcf&       u,                     // matrix of computed left vectors U
   matrixcf&       v,                     // matrix of computed left vectors V
   vectorf&        work_results           // additional computation results
   );
```

**Parameters**

*joba*

[in] ENUM_SVDJH_A enumeration value determining the accuracy level of the SVD computation.

*jobu*

[in] ENUM_SVDJH_U enumeration value that determines how the left singular vectors should be computed.

*jobv*

[in] ENUM_SVDJH_V enumeration value that determines how the right singular vectors should be computed.

*jobr*

[in] ENUM_SVDJH_R enumeration value defining the range of computable values

*jobt*

[in] ENUM_SVDJH_T enumeration value defining whether to transpose the matrix of it is square. If the matrix is non-square, this parameter is ignored.

*jobp*

[in] ENUM_SVDJH_P enumeration value defining the possibility of structured perturbations to remove denormalized values.

*S*

[out] Vector of singular values.

For work(1)/work(2) = one: the singular values of A. During the computation S contains Euclidean column norms of the iterated matrices in the array a.

For work(1)≠work(2): the singular values of A are (work(1)/work(2)) * S(1:n). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A.

jobr = 'R', some of the singular values may be returned as exact zeros obtained by 'setting to zero' because they are below the numerical rank threshold or are denormalized numbers.

*U*

© 2000-2025, MetaQuotes Ltd.

[out] Matrix of left singular vectors.

*V*

[out] Matrix of right singular vectors.

*work_results*

[out] Vector consisting of 7 statistics obtained as a result of the computation.

work(1) = scale = work(2)/work(1) is the scaling factor such that scale*sva(1:n) are the computed singular values of A. See the description of S.

work(2) = see the description of work(1).

work(3) = sconda is an estimate for the condition number of column equilibrated A. If joba = 'E' or 'G', sconda is an estimate of sqrt(||(R**t * R)**(-1)||_1). It is computed using ?pocon. It holds n**(-1/4) * sconda ≤ ||R**(-1)||_2 ≤ n**(1/4) * sconda, where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than n, sconda is returned as -1, indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a user who is familiar with the details of the method.

work(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization.

work(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization.

The following two parameters are computed if jobt = 'T'. They are provided for a user who is familiar with the details of the method.

work(6) = the entropy of A**t*A :: this is the Shannon entropy of diag(A**t*A) / Trace(A**t*A) taken as point in the probability simplex.

work(7) = the entropy of A*A**t.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The number of matrix rows must not be less than the number of columns.

**ENUM_SVDJH_A**

An enumeration that specifies the level of accuracy of the SVD computation.

| ID | Description |
|---|---|
| SVDJHA_C | Computation as with 'C' with an additional estimate of the condition number. It provides a realistic error bound. |
| SVDJHA_E | Computation as with SVDJHA_C' with an additional estimate of the condition number. It provides a realistic error bound. |

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDJHA_F | Higher accuracy than the SVDJHA_C option. |
| SVDJHA_G | Computation as with SVDJHA_F with an additional estimate of the condition number. |
| SVDJHA_A | Small singular values are the noise and the matrix is treated as numerically rank deficient. |
| SVDJHA_R | Similar as in SVDJHA_A, but more accuracy. |

**ENUM_SVDJH_U**

An enumeration defining how left singular vectors should be computed.

| ID | Description |
| --- | --- |
| SVDJHU_U | N columns of U are returned in the array U. |
| SVDJHU_F | Full set of M left singular vectors is returned in the array U. |
| SVDJHU_N | U is not computed. |

**ENUM_SVDJH_V**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
| --- | --- |
| SVDJHV_V | N columns of V are returned in the array V |
| SVDJHV_J | N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations |
| SVDJHV_N | V is not computed |

**ENUM_SVDJH_R**

An enumeration that defines the range of values to be computed.

| ID | Description |
| --- | --- |
| SVDJHR_N | Do not kill small columns of c*A. |
| SVDJHR_R | RESTRICTED range for sigma(c*A). This option is recommended. |

**ENUM_SVDJH_T**

An enumeration that specifies whether a matrix should be transposed if it is square.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDJHT_T | Transpose if entropy test indicates possibly faster convergence of Jacobi process. |
| SVDJHT_N | Do not use transposition. Do not speculate. |

**ENUM_SVDJH_P**

An enumeration that specifies the possibility of structured perturbations to remove denormalized values.

| ID | Description |
| --- | --- |
| SVDJHP_P | Introduce perturbation. |
| SVDJHP_N | Do not perturb. |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQR

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionJacobiLow

Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases.

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vector&        S,               // vector of computed singular values
   matrix&        U,               // matrix of computed left vectors U
   matrix&        V,               // matrix of computed left vectors V
   vector&        work_results     // additional computation results
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vectorf&       S,               // vector of computed singular values
   matrixf&       U,               // U matrix of computed left vectors
   matrixf&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vector&        S,               // vector of computed singular values
   matrixc&       U,               // U matrix of computed left vectors
   matrixc&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U    jobu,                  // how to compute left vectors
   ENUM_SVDJH_V    jobv,                  // how to compute right vectors
```

© 2000-2025, MetaQuotes Ltd.

```
   double          ctol,                  // threshold for convergence if jobu='C'
   ulong           mv,                    // number of first rows of matrix V if jobv
   vectorf&        singular_values,       // vector of computed singular values S
   matrixcf&       u,                     // matrix of computed left vectors U
   matrixcf&       v,                     // matrix of computed left vectors V
   vectorf&        work_results           // additional computation results
   );
```

**Parameters**

*jobu*

[in] ENUM_SVDJL_U enumeration value that determines how the left singular vectors should be computed.

*jobv*

[in] ENUM_SVDJL_V enumeration value defining how the right singular vectors should be computed.

*ctol*

[in] Convergence threshold if jobu=SVDJLU_C. For other values of 'jobu' the parameter is ignored.

*mv*

[in] Number of rows of matrix V to be computed if jobv=SVDJLV_A. For other values of 'jobv' the parameter is ignored.

*S*

[out] Vector of singular values.

Depending on the value scale = work(1), where scale is the scaling factor:

if scale = 1, S(1:n) contains the computed singular values of a. During the computation, sva contains the Euclidean column norms of the iterated matrices in the array a.

if scale ≠ 1, the singular values of a are scale*S(1:n), and this factored representation is due to the fact that some of the singular values of a might underflow or overflow.

*U*

[out] Matrix of left singular vectors.

*V*

[out] Matrix of right singular vectors (non-transposed).

*work_results*

[out] Vector consisting of 7 statistics obtained as a result of the computation.

work(1) = scale is the scaling factor such that scale*S(1:n) are the computed singular values of A. See the description of S).

work(2) is the number of the computed nonzero singular value.

work(3) is the number of the computed singular values that are larger than the underflow threshold.

work(4) is the number of sweeps of Jacobi rotations needed for numerical convergence.

© 2000-2025, MetaQuotes Ltd.

work(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when ?gesvj did not converge, as it can be used to estimate whether the output is still useful and for post festum analysis.

work(6) is the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful in a post festum analysis.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The number of matrix rows must not be less than the number of columns.

**ENUM_SVDJL_U**

An enumeration defining how left singular vectors should be computed.

| ID | Description |
|---|---|
| SVDJLU_U | The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A |
| SVDJLU_C | Analogous to SVDJLU_U, except that user can control the level of numerical orthogonality of the computed left singular vectors. |
| SVDJLU_N | The matrix U is not computed. |

**ENUM_SVDJL_V**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| SVDJLV_V | The matrix V is computed |
| SVDJLV_A | The Jacobi rotations are applied to the MV-by-N matrix V. |
| SVDJLV_N | The matrix V is not computed. |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQR

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionBidiagDC

Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSDC).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagDC(
   ENUM_SVDBIDIAG_Z    jobz,        // how to compute left vectors
   ENUM_BLAS_RANGE     range,       // subset of computed singular values
   double              lower,       // lower limit of the subset
   double              upper,       // upper limit of the subset
   vector&             S,           // vector of computed singular values
   matrix&             U,           // U matrix of computed left vectors
   matrix&             VT           // VT transposed matrix of right vectors
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagDC(
   ENUM_SVDBIDIAG_Z    jobz,        // how to compute left vectors
   ENUM_BLAS_RANGE     range,       // subset of computed singular values
   double              lower,       // lower limit of the subset
   double              upper,       // upper limit of the subset
   vectorf&            S,           // vector of computed singular values
   matrixf&            U,           // U matrix of computed left vectors
   matrixf&            VT           // VT transposed matrix of right vectors
   );
```

### Parameters

*jobz*

[in] ENUM_SVDBIDIAG_Z enumeration value that determines how the left singular vectors should be computed.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

*lower*

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

*upper*

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

© 2000-2025, MetaQuotes Ltd.

*V*

[out] Transposed matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the jobz and range parameters.

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

A bidiagonal matrix is a square matrix with non-zero main diagonal and one of the sub-diagonals.

**Upper bidiagonal matrix**

```
[[x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x],
 [0, 0, 0, 0, x]]
```

**Lower bidiagonal matrix**

```
[[x, 0, 0, 0, 0],
 [x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x]]
```

**ENUM_SVDBIDIAG_Z**

An enumeration defining how left singular vectors should be computed.

| ID | Description |
| --- | --- |
| SVDJOBZ_V | Compute singular values and singular vectors. |
| SVDJOBZ_N | Compute singular values only. |

**ENUM_BLAS_RANGE**

© 2000-2025, MetaQuotes Ltd.

An enumeration defining how right singular vectors should be computed.

| ID | Description |
| --- | --- |
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

**See also**

SingularValueDecompositionDC, SingularValueDecompositionQR

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionBidiagBisect

Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX).

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagBisect(
   ENUM_SVDBIDIAG_Z    jobz,       // how to compute left vectors
   ENUM_BLAS_RANGE     range,      // subset of computed singular values
   double              lower,      // lower limit of the subset
   double              upper,      // upper limit of the subset
   vector&             S,          // vector of computed singular values
   matrix&             U,          // U matrix of computed left vectors
   matrix&             VT          // VT transposed matrix of right vectors
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagBisect(
   ENUM_SVDBIDIAG_Z    jobz,       // how to compute left vectors
   ENUM_BLAS_RANGE     range,      // subset of computed singular values
   double              lower,      // lower limit of the subset
   double              upper,      // upper limit of the subset
   vectorf&            S,          // vector of computed singular values
   matrixf&            U,          // U matrix of computed left vectors
   matrixf&            VT          // VT transposed matrix of right vectors
   );
```

### Parameters

*jobz*

[in] ENUM_SVDBIDIAG_Z enumeration value that determines how the left singular vectors should be computed.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

*lower*

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

*upper*

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

© 2000-2025, MetaQuotes Ltd.

*V*

[out] Transposed matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

A bidiagonal matrix is a square matrix with non-zero main diagonal and one of the sub-diagonals.

**Upper bidiagonal matrix**

```
[[x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x],
 [0, 0, 0, 0, x]]
```

**Lower bidiagonal matrix**

```
[[x, 0, 0, 0, 0],
 [x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x]]
```

**ENUM_SVDBIDIAG_Z**

An enumeration defining how left singular vectors should be computed.

| ID | Description |
| --- | --- |
| SVDJOBZ_V | Compute singular values and singular vectors. |
| SVDJOBZ_N | Compute singular values only. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

© 2000-2025, MetaQuotes Ltd.

# SingularValueDecompositionBidiagQR

Computes the singular value decomposition of a general matrix that has been reduced to bidiagonal form by the method ReduceToBidiagonal. Lapack function BDSQR.

### Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,          // orthogonal matrix Q
   matrix&          PT,         // transposed matrix P
   vector&          S,          // vector of computed singular values
   matrix&          U,          // matrix U of computed left vectors
   matrix&          VT          // transposed matrix V of right vectors
   );
```

### Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,          // orthogonal matrix Q
   matrix&          PT,         // transposed matrix P
   vectorf&         S,          // vector of computed singular values
   matrixf&         U,          // matrix U of computed left vectors
   matrixf&         VT          // transposed matrix V of right vectors
   );
```

### Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,          // orthogonal matrix Q
   matrix&          PH,         // Hermitian conjugated matrix P
   vector           S,          // vector of computed singular values
   matrixc          U,          // matrix U of computed left vectors
   matrixc          VH          // Hermitian conjugated matrix V of right vectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrixf&         Q,          // orthogonal matrix Q
   matrixf&         PH,         // Hermitian conjugated matrix P
   vectorf&         S,          // vector of computed singular values
   matrixcf&        U,          // matrix U of computed left vectors
   matrixcf&        VH          // Hermitian conjugated matrix V of right vectors
   );
```

### Parameters

*Q*

[in] Orthogonal matrix Q produced by method ReflectBidiagonalToQP. If matrix Q has zero size, then left vectors U are not calculated.

*PT*

© 2000-2025, MetaQuotes Ltd.

[in] Transposed (or hermitian conjugated) matrix P produced by method ReflectBidiagonalToQP. If matrix PT has zero size, then right vectors VT are not calculated.

*S*

[out] Vector of singular values.

*U*

[out] Matrix of left singular vectors.

*VT*

[out] Matrix of right singular vectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

BDSQR computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

B = Q * S * P**T

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P**T*VT instead of P**T, for given real input matrices U and VT. When U and VT are the orthogonal matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by GEBRD, then

A = (U*Q) * S * (P**T*VT)

is the SVD of A.

© 2000-2025, MetaQuotes Ltd.

# Eigen Values

The section features functions for computing eigenvalues and eigenvectors. It describes methods for solving standard linear algebra problems using the LAPACK library algorithms. These functions are efficient for matrix analysis, diagonalization, system stabilization, and other tasks.

- **EigenSolver**: The function is designed to compute the eigenvalues and eigenvectors of an arbitrary square matrix using the classical algorithm represented by the GEEV lapack function. This method is applied to a wide range of matrices, allowing the decomposition of matrices into their eigenvalues and eigenvectors.
- **EigenSymmetricDC**: The function for computing eigenvalues and eigenvectors of symmetric or Hermitian matrices using the divide-and-conquer algorithm. The lapack functions SYEVD and HEEVD enable the efficient handling of symmetric or Hermitian matrices, providing faster and more accurate processing of such matrices.

| Function | Action |
| --- | --- |
| EigenSolver | Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV). |
| EigenSolverX | Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX). |
| EigenSolverShur | Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition. |
| EigenSolver2 | Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV). |
| EigenSolver2X | Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size. |
| EigenSolver2Shur | Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES). |
| EigenSolver2Blocked | Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2. |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| EigenSolver2ShurBlocked | Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3). |
| EigenSymmetricDC | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD). |
| EigenSymmetricQR | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV). |
| EigenSymmetricRobust | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR). |
| EigenSymmetricBisect | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX). |

© 2000-2025, MetaQuotes Ltd.

# General Matrices

Functions for calculating eigenvalues and eigenvectors of a square matrix using classical algorithms. It provides various methods for working with both real and complex matrices, allowing you to solve linear algebra problems with a choice of methods for calculating eigenvectors.

| Function | Action |
| --- | --- |
| EigenSolver | Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV). |
| EigenSolverX | Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX). |
| EigenSolverShur | Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition. |
| EigenSolver2 | Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV). |
| EigenSolver2X | Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size. |
| EigenSolver2Shur | Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES). |
| EigenSolver2Blocked | Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2. |
| EigenSolver2ShurBlocked | Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3). |

© 2000-2025, MetaQuotes Ltd.

# EigenSolver

Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV).

### Computing for type matrix<double>

```
bool  matrix::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                 // compute left and right eigenvecto
   vectorc&              eigen_values,         // vector of computed eigenvalues
   matrix&               left_eigenvectors,    // matrix of computed left vectors
   matrix&               right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                 // compute left and right eigenvecto
   vectorcf&             eigen_values,         // vector of computed eigenvalues
   matrixf&              left_eigenvectors,    // matrix of computed left vectors
   matrixf&              right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                 // compute left and right vectors
   vectorc&              eigen_values,         // vector of computed eigenvalues
   matrixc&              left_eigenvectors,    // matrix of computed left vectors
   matrixc&              right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                 // compute left and right vectors
   vectorcf&             eigen_values,         // vector of computed eigenvalues
   matrixcf&             left_eigenvectors,    // matrix of computed left vectors
   matrixcf&             right_eigenvectors    // matrix of computed right vectors
   );
```

### Parameters

*jobv*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*EV*

[out] Vector of eigenvalues.

*left_eigenvectors*

[out] Matrix of left eigenvectors.

© 2000-2025, MetaQuotes Ltd.

*right_eigenvectors*

[out] Matrix of right eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobv parameter.

If EIGVECTORS_N is set, the left and right vectors are not computed. Only eigenvalues are computed.

With EIGVECTORS_L, only left eigenvectors are computed, right eigenvectors are not computed.

When EIGVECTORS_R is set, only the right eigenvectors are computed, the left vectors are not computed.

With EIGVECTORS_LR, the left and right eigenvectors are computed, Eigenvalues are always computed.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

**ENUM_EIG_VECTORS**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
|---|---|
| EIGVECTORS_N | Only eigenvalues are calculated, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

© 2000-2025, MetaQuotes Ltd.

# EigenSolverX

Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX).

**Computing for type matrix<double>**

```
bool  matrix::EigenSolverX(
   ENUM_EIG_BALANCE      balance,              // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                 // determines computation of right a
   ENUM_EIG_SENSE        sense,                // determines computation of recipr
   vectorc&              eigen_values,         // vector of computed eigenvalues
   matrix&               left_eigenvectors,    // matrix of computed left vectors
   matrix&               right_eigenvectors    // matrix of computed right vectors
   matrix&               shur_matrix,          // balanced matrix in Schur form
   long&                 ilo,                  // subscript of balanced matrix
   long&                 ihi,                  // superscript of balanced matrix
   vector&               scale,                // details of permutations and scali
   double&               ab_norm,              // 1-norm of balanced matrix
   vector&               rconde,               // vector of reciprocal condition nu
   vector&               rcondv                // vector of reciprocal condition nu
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSolverX(
   ENUM_EIG_BALANCE      balance,              // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                 // determines computation of right a
   ENUM_EIG_SENSE        sense,                // determines computation of recipr
   vectorcf&             eigen_values,         // vector of computed eigenvalues
   matrixf&              left_eigenvectors,    // matrix of computed left vectors
   matrixf&              right_eigenvectors    // matrix of computed right vectors
   matrixf&              shur_matrix,          // balanced matrix in Schur form
   long&                 ilo,                  // subscript of balanced matrix
   long&                 ihi,                  // superscript of balanced matrix
   vectorf&              scale,                // details of permutations and scali
   float&                ab_norm,              // 1-norm of balanced matrix
   vectorf&              rconde,               // vector of reciprocal condition nu
   vectorf&              rcondv                // vector of reciprocal condition nu
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSolverX(
   ENUM_EIG_BALANCE      balance,              // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                 // determines computation of right a
   ENUM_EIG_SENSE        sense,                // determines computation of recipr
   vectorc&              eigen_values,         // vector of computed eigenvalues
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixc&              left_eigenvectors,       // matrix of computed left vectors
   matrixc&              right_eigenvectors       // matrix of computed right vectors
   matrixc&              shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vector&               scale,                   // details of permutations and scali
   double&               ab_norm,                 // 1-norm of balanced matrix
   vector&               rconde,                  // vector of reciprocal condition nu
   vector&               rcondv                   // vector of reciprocal condition nu
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSolverX(
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixcf&             left_eigenvectors,       // matrix of computed left vectors
   matrixcf&             right_eigenvectors       // matrix of computed right vectors
   matrixcf&             shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vectorf&              scale,                   // details of permutations and scali
   float&                ab_norm,                 // 1-norm of balanced matrix
   vectorf&              rconde,                  // vector of reciprocal condition nu
   vectorf&              rcondv                   // vector of reciprocal condition nu
   );
```

**Parameters**

*balance*

[in] Value from the ENUM_EIG_BALANCE enumeration which determines the need and method for balancing the input matrix; it is used to improve the conditioning of the eigenvalues and eigenvectors.

*jobv*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*sense*

[in] Value from the ENUM_EIG_SENSE enumeration determining the need to compute reciprocal condition numbers.

*eigen_values*

[out] Vector of eigenvalues.

*left_eigenvectors*

[out] Matrix of left eigenvectors.

*right_eigenvectors*

© 2000-2025, MetaQuotes Ltd.

[out] Matrix of right eigenvectors.

*shur_matrix*

[out] Balanced matrix in Schur form; the matrix is not filled if neither left nor right eigenvectors are computed.

*ilo*

[out] Subscript of the balanced matrix; the matrix is not filled if no balancing is applied.

*ihi*

[out] Superscript of the balanced matrix; the matrix is not filled if no balancing is applied.

*scale*

[out] Vector of details of permutations and scaling when balancing the input matrix.

Details of the permutations and scaling factors applied when balancing A.

If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then

scale(j) = P(j), for j = 1,...,ilo-1

= D(j), for j = ilo,...,ihi

= P(j) for j = ihi+1,..., n.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

*ab_norm*

[out] 1-norm of the balanced matrix (the maximum of the sum of absolute values of elements in any of the matrix columns).

*rconde*

[out] Vector of reciprocal condition numbers for each eigenvalue; it is computed if the 'sense' parameter is set to 'E' or 'B'.

*rcondv*

[out] Vector of reciprocal condition numbers for each eigenvector; it is computed if the 'sense' parameter is set to 'V' or 'B.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Input matrix balancing depends on the value of the '*balance*' parameter.

**ENUM_EIG_BALANCE**

An enumeration defining the need to compute eigenvectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGBALANCE_N | Do not diagonally scale or permute |
| EIGBALANCE_P | Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale |
| EIGBALANCE_S | Diagonally scale the matrix. Do not permute |
| EIGBALANCE_B | Both diagonally scale and permute |

### ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

| ID | Description |
| --- | --- |
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

### ENUM_EIG_SENSE

An enumeration defining the need to compute eigenvectors.

| ID | Description |
| --- | --- |
| EIGSENSE_N | None of reciprocal condition numbers are computed |
| EIGSENSE_E | Computed for eigenvalues only |
| EIGSENSE_V | Computed for right eigenvectors only |
| EIGSENSE_B | Computed for eigenvalues and right eigenvectors |

© 2000-2025, MetaQuotes Ltd.

# EigenSolverShur

Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition.

**Computing for type matrix<double>**

```
bool  matrix::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                  // method to compute Shur vectors
   vectorc&              eigen_values,           // vector of computed eigenvalues
   matrix&               shur_matrix,            // matrix in Schur form
   matrix&               shur_vectors            // matrix of Schur vectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                  // method to compute Shur vectors
   vectorcf&             eigen_values,           // vector of computed eigenvalues
   matrixf&              shur_matrix,            // matrix in Schur form
   matrixf&              shur_vectors            // matrix of Schur vectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                  // method to compute Shur vectors
   vectorc&              eigen_values,           // vector of computed eigenvalues
   matrixc&              shur_matrix,            // matrix in Schur form
   matrixc&              shur_vectors            // matrix of Schur vectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                  // method to compute Shur vectors
   vectorcf&             eigen_values,           // vector of computed eigenvalues
   matrixcf&             shur_matrix,            // matrix in Schur form
   matrixcf&             shur_vectors            // matrix of Schur vectors
   );
```

**Parameters**

*jobvs*

[in] Value from the ENUM_EIG_SHUR enumeration, which defines the method for computing Shur vectors.

*eigen_values*

[out] Vector of eigenvalues.

*shur_matrix*

[out] Upper triangular Schur matrix (Schur form for the input matrix).

© 2000-2025, MetaQuotes Ltd.

*shur_vectors*

[out] Matrix of Schur vectors; it is not computed if the *jobvs* parameter is set to N.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the *jobvs* parameter values.

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

**EigenSolverShur**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGSHUR_N | Schur vectors are not computed |
| EIGSHUR_V | Schur vectors are computed |

© 2000-2025, MetaQuotes Ltd.

# EigenSolver2

Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV). Both matrices must be the same size.

### Computing for type matrix<double>

```
bool  matrix::EigenSolver2(
   matrix&               B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorc&              alpha,                // vector of computed eigenvalues
   vector&               beta,                 // vector of eigenvalue divisors
   matrix&               left_eigenvectors,    // matrix of computed left vectors
   matrix&               right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSolver2(
   matrix&               B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorcf&             alpha,                // vector of computed eigenvalues
   vectorf&              beta,                 // vector of eigenvalue divisors
   matrixf&              left_eigenvectors,    // matrix of computed left vectors
   matrixf&              right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSolver2(
   matrixc&              B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorc&              alpha,                // vector of computed eigenvalues
   vectorc&              beta,                 // vector of eigenvalue divisors
   matrixc&              left_eigenvectors,    // matrix of computed left vectors
   matrixc&              right_eigenvectors    // matrix of computed right vectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2(
   matrixcf&             B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorcf&             alpha,                // vector of computed eigenvalues
   vectorcf&             beta,                 // vector of eigenvalue divisors
   matrixcf&             left_eigenvectors,    // matrix of computed left vectors
   matrixcf&             right_eigenvectors    // matrix of computed right vectors
   );
```

### Parameters

© 2000-2025, MetaQuotes Ltd.

*B*

[in] The second matrix in the pair.

*jobv*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*alpha*

[out] Vector of eigenvalues.

*beta*

[out] Vector of eigen value divisors.

*left_eigenvectors*

[out] Matrix of left eigenvectors.

*right_eigenvectors*

[out] Matrix of right eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobv parameter.

**ENUM_EIG_VECTORS**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

A * v(j) = lambda(j) * B * v(j).

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

u(j)**H * A = lambda(j) * u(j)**H * B .

© 2000-2025, MetaQuotes Ltd.

where u(j)**H is the conjugate-transpose of u(j).

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

© 2000-2025, MetaQuotes Ltd.

# EigenSolver2X

Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size.

Optionally, it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

**Computing for type matrix<double>**

```
bool  matrix::EigenSolver2X(
   matrix&               B,                    // second matrix in the pair
   ENUM_EIG_BALANCE      balance,              // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                 // determines computation of right a
   ENUM_EIG_SENSE        sense,                // determines computation of recipro
   vectorc&              alpha,                // vector of computed eigenvalues
   vector&               beta,                 // vector of eigenvalue divisors
   matrix&               left_eigenvectors,    // matrix of computed left vectors
   matrix&               right_eigenvectors    // matrix of computed right vectors
   matrix&               shur_matrix1,         // the first part of the real Schur
   matrix&               shur_matrix2,         // the second part of the real Schu
   long&                 ilo,                  // subscript of balanced matrix
   long&                 ihi,                  // superscript of balanced matrix
   vector&               lscale,               // details of the permutations and s
   vector&               rscale,               // details of the permutations and s
   double&               ab_norm,              // one-norm of balanced input matri
   double&               bb_norm,              // one-norm of balanced second matr
   vector&               rconde,               // vector of reciprocal condition nu
   vector&               rcondv                // vector of reciprocal condition nu
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSolver2X(
   matrixf&              B,                    // second matrix in the pair
   ENUM_EIG_BALANCE      balance,              // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                 // determines computation of right a
   ENUM_EIG_SENSE        sense,                // determines computation of recipro
   vectorcf&             alpha,                // vector of computed eigenvalues
   vectorf&              beta,                 // vector of eigenvalue divisors
   matrixf&              left_eigenvectors,    // matrix of computed left vectors
   matrixf&              right_eigenvectors    // matrix of computed right vectors
   matrixf&              shur_matrix1,         // the first part of the real Schur
   matrixf&              shur_matrix2,         // the second part of the real Schu
   long&                 ilo,                  // subscript of balanced matrix
   long&                 ihi,                  // superscript of balanced matrix
   vectorf&              lscale,               // details of the permutations and s
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf&              rscale,                 // details of the permutations and s
   float&                ab_norm,                // one-norm of balanced input matrix
   float&                bb_norm,                // one-norm of balanced second matri
   vectorf&              rconde,                 // vector of reciprocal condition nu
   vectorf&              rcondv                  // vector of reciprocal condition nu
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolver2X(
   matrixc&              B,                      // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                   // determines computation of right a
   ENUM_EIG_SENSE        sense,                  // determines computation of recipro
   vectorc&              alpha,                  // vector of computed eigenvalues
   vectorc&              beta,                   // vector of eigenvalue divisors
   matrixc&              left_eigenvectors,      // matrix of computed left vectors
   matrixc&              right_eigenvectors      // matrix of computed right vectors
   matrixc&              shur_matrix1,           // the first part of the real Schur
   matrixc&              shur_matrix2,           // the second part of the real Schur
   long&                 ilo,                    // subscript of balanced matrix
   long&                 ihi,                    // superscript of balanced matrix
   vector&               lscale,                 // details of the permutations and s
   vector&               rscale,                 // details of the permutations and s
   double&               ab_norm,                // one-norm of balanced input matrix
   double&               bb_norm,                // one-norm of balanced second matri
   vector&               rconde,                 // vector of reciprocal condition nu
   vector&               rcondv                  // vector of reciprocal condition nu
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2X(
   matrixcf&             B,                      // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                   // determines computation of right a
   ENUM_EIG_SENSE        sense,                  // determines computation of recipro
   vectorcf&             alpha,                  // vector of computed eigenvalues
   vectorf&              beta,                   // vector of eigenvalue divisors
   matrixcf&             left_eigenvectors,      // matrix of computed left vectors
   matrixcf&             right_eigenvectors      // matrix of computed right vectors
   matrixcf&             shur_matrix1,           // the first part of the real Schur
   matrixcf&             shur_matrix2,           // the second part of the real Schur
   long&                 ilo,                    // subscript of balanced matrix
   long&                 ihi,                    // superscript of balanced matrix
   vectorf&              lscale,                 // details of the permutations and s
   vectorf&              rscale,                 // details of the permutations and s
   float&                ab_norm,                // one-norm of balanced input matrix
```

© 2000-2025, MetaQuotes Ltd.

```
   float&               bb_norm,              // one-norm of balanced second matri
   vectorf&             rconde,               // vector of reciprocal condition nu
   vectorf&             rcondv                // vector of reciprocal condition nu
   );
```

**Parameters**

*B*

[in] The second matrix in the pair.

*balance*

[in] Value from the ENUM_EIG_BALANCE enumeration which determines the need and method for balancing the input matrix; it is used to improve the conditioning of the eigenvalues and eigenvectors.

*jobv*

[in] Value from the ENUM_EIG_VECTORS enumeration which determines the method for computing left and right eigenvectors.

*sense*

[in] Value from the ENUM_EIG_SENSE enumeration which determines the need to compute reciprocal condition numbers.

*eigen_values*

[out] Vector of eigenvalues.

*left_eigenvectors*

[out] Matrix of left eigenvectors.

*right_eigenvectors*

[out] Matrix of right eigenvectors.

*shur_matrix1, shur_matrix2*

[out] 2 parts of balanced matrix in Schur form; the matrix is not filled if neither left nor right eigenvectors are computed.

*ilo*

[out] Subscript of the balanced matrix; the matrix is not filled if no balancing is applied.

*ihi*

[out] Superscript of the balanced matrix; the matrix is not filled if no balancing is applied.

*lscale*

[out] Vector contains details of the permutations and scaling factors applied to the left side of A and B.

If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then

lscale(j) = PL(j), for j = 1,..., ilo-1

= DL(j), for j = ilo,...,ihi

= PL(j) for j = ihi+1,..., n.

© 2000-2025, MetaQuotes Ltd.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

*rscale*

[out] Vector contains details of the permutations and scaling factors applied to the right side of A and B.

If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then

rscale(j) = PR(j), for j = 1,..., ilo-1

= DR(j), for j = ilo,...,ihi

= PR(j) for j = ihi+1,..., n.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

*ab_norm*

[out] One-norm of the balanced input matrix (the maximum of the sum of absolute values of elements in any of the matrix columns).

*bb_norm*

[out] One-norm of balanced second matrix B.

*rconde*

[out] Vector of reciprocal condition numbers for each eigenvalue; it is computed if the 'sense' parameter is set to 'E' or 'B'.

*rcondv*

[out] Vector of reciprocal condition numbers for each eigenvector; it is computed if the 'sense' parameter is set to 'V' or 'B'.

**Return Value**

The function returns 'true' on success or 'false' if an error occurs.

**Note**

Input matrices balancing depends on the value of the *balance* parameter.

**ENUM_EIG_BALANCE**

An enumeration that specifies whether the matrices should be balanced.

| ID | Description |
|---|---|
| EIGBALANCE_N | Do not diagonally scale or permute. |
| EIGBALANCE_P | Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale. |
| EIGBALANCE_S | Diagonally scale the matrix. Do not permute. |
| EIGBALANCE_B | Both diagonally scale and permute. |

© 2000-2025, MetaQuotes Ltd.

### ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

### ENUM_EIG_SENSE

An enumeration determining the need to compute reciprocal condition numbers.

| ID | Description |
|---|---|
| EIGSENSE_N | None of reciprocal condition numbers are computed. |
| EIGSENSE_E | Computed for eigenvalues only. |
| EIGSENSE_V | Computed for right eigenvectors only. |
| EIGSENSE_B | Computed for eigenvalues and right eigenvectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSolver2Shur

Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES).

Computes the generalized eigenvalues, the generalized real/complex Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (vsl and vsr) for a pair of n-by-n real/complex nonsymmetric matrices (A,B). This gives the generalized Schur factorization:

(A,B) = ( vsl*S *vsrH, vsl*T*vsrH )

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T. The leading columns of vsl and vsr then form an orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

**Computing for type matrix<double>**

```
bool  matrix::EigenSolver2Shur(
   matrix&                 B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vector&                 alpha,     // vector of computed eigenvalues
   vector&                 beta,      // vector of eigenvalue divisors
   matrix&                 shur_s,    // matrix S in Schur form
   matrix&                 shur_t,    // matrix T in Schur form
   matrix&                 vsl,       // matrix of left Schur vectors vsl
   matrix&                 vsr        // matrix of right Schur vectors vsr
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSolver2Shur(
   matrixf&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorcf&               alpha,     // vector of computed eigenvalues
   vectorf&                beta,      // vector of eigenvalue divisors
   matrixf&                shur_s,    // matrix S in Schur form
   matrixf&                shur_t,    // matrix T in Schur form
   matrixf&                vsl,       // matrix of left Schur vectors vsl
   matrixf&                vsr        // matrix of right Schur vectors vsr
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::EigenSolver2Shur(
   matrixc&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorc&                alpha,     // vector of computed eigenvalues
   vectorc&                beta,      // vector of eigenvalue divisors
   matrixc&                shur_s,    // matrix S in Schur form
   matrixc&                shur_t,    // matrix T in Schur form
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixc&                vsl,       // matrix of left Schur vectors vsl
   matrixc&                vsr        // matrix of right Schur vectors vsr
   );
```

**Вычисления для типа matrix<complexf>**

```
bool  matrixcf::EigenSolver2Shur(
   matrixcf&               B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorcf&               alpha,     // vector of computed eigenvalues
   vectorcf&               beta,      // vector of eigenvalue divisors
   matrixcf&               shur_s,    // matrix S in Schur form
   matrixcf&               shur_t,    // matrix T in Schur form
   matrixcf&               vsl,       // matrix of left Schur vectors vsl
   matrixcf&               vsr        // matrix of right Schur vectors vsr
   );
```

**Parameters**

*B*

[out] The second matrix in the pair.

*jobvs*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*alpha*

[out] Vector of eigenvalues.

*beta*

[out] Vector of eigenvalue divisors.

*shur_s*

[out] Matrix S, block upper triangular Schur matrix (Schur form for the input matrix).

*shur_t*

[out] Matrix T, block upper triangular Schur matrix (Schur form for the second matrix in the pair).

*vsl*

[out] Matrix of left Schur vectors.

*vsr*

[out] Matrix of right Schur vectors.

**Return Value**

The function returns 'true' on success or 'false' if an error occurs.

**Note**

Computation depends on the *jobvs* parameter values.

© 2000-2025, MetaQuotes Ltd.

The second matrix in the pair must be the same size as the first (input) one.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

### EigenSolverShur

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

© 2000-2025, MetaQuotes Ltd.

# EigenSolver2Blocked

Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2.

**Computing for type matrix<double>**

```
bool  matrix::EigenSolver2Blocked(
   matrix&               B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorc&              alpha,                // vector of computed eigenvalues
   vector&               beta,                 // vector of eigenvalue divisors
   matrix&               left_eigenvectors,    // matrix of computed left vectors
   matrix&               right_eigenvectors    // matrix of computed right vectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSolver2Blocked(
   matrix&               B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorcf&             alpha,                // vector of computed eigenvalues
   vectorf&              beta,                 // vector of eigenvalue divisors
   matrixf&              left_eigenvectors,    // matrix of computed left vectors
   matrixf&              right_eigenvectors    // matrix of computed right vectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::EigenSolver2Blocked(
   matrixc&              B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorc&              alpha,                // vector of computed eigenvalues
   vectorc&              beta,                 // vector of eigenvalue divisors
   matrixc&              left_eigenvectors,    // matrix of computed left vectors
   matrixc&              right_eigenvectors    // matrix of computed right vectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSolver2(
   matrixcf&             B,                    // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                 // method to compute right and left
   vectorcf&             alpha,                // vector of computed eigenvalues
   vectorcf&             beta,                 // vector of eigenvalue divisors
   matrixcf&             left_eigenvectors,    // matrix of computed left vectors
   matrixcf&             right_eigenvectors    // matrix of computed right vectors
   );
```

**Parameters**

© 2000-2025, MetaQuotes Ltd.

*B*

[out] The second matrix in the pair.

*jobv*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*alpha*

[out] Vector of eigenvalues.

*beta*

[out] Vector of eigenvalue divisors.

*left_eigenvectors*

[out] Matrix of left eigenvectors.

*righeft_eigenvectors*

[out] Matrix of right eigenvectors.

**Return Value**

The function returns 'true' on success or 'false' if an error occurs.

**Note**

Computation depends on the value of the jobv parameter.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies:

A * v(j) = lambda(j) * B * v(j).

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies:

u(j)**H * A = lambda(j) * u(j)**H * B .

where u(j)**H is the conjugate-transpose of u(j).

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

**ENUM_EIG_VECTORS**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
| --- | --- |
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

© 2000-2025, MetaQuotes Ltd.

# EigenSolver2ShurBlocked

Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3).

Computes the generalized eigenvalues, the generalized real/complex Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR) for a pair of n-by-n real/complex nonsymmetric matrices (A,B). This gives the generalized Schur factorization:

(A,B) = ( vsl*S *vsrH, vsl*T*vsrH )

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T. The leading columns of vsl and vsr then form an orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

**Computing for type matrix<double>**

```
bool  matrix::EigenSolver2ShurBlocked(
   matrix&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS       jobvs,     // method to compute left and right vectors
   vectorc&               alpha,     // vector of computed eigenvalues
   vector&                beta,      // vector of eigenvalue divisors
   matrix&                shur_s,    // matrix S in Schur form
   matrix&                shur_t,    // matrix T in Schur form
   matrix&                vsl,       // matrix of left Schur vectors VSL
   matrix&                vsr        // matrix of right Schur vectors VSR
   );
```

**Computing for type matrix<float>**

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixf&               B,         // second matrix in the pair
   ENUM_EIG_VECTORS       jobvs,     // method to compute left and right vectors
   vectorcf&              alpha,     // vector of computed eigenvalues
   vectorf&               beta,      // vector of eigenvalue divisors
   matrixf&               shur_s,    // matrix S in Schur form
   matrixf&               shur_t,    // matrix T in Schur form
   matrixf&               vsl,       // matrix of left Schur vectors VSL
   matrixf&               vsr        // matrix of right Schur vectors VSR
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixc&               B,         // second matrix in the pair
   ENUM_EIG_VECTORS       jobvs,     // method to compute left and right vectors
   vectorc&               alpha,     // vector of computed eigenvalues
   vectorc&               beta,      // vector of eigenvalue divisors
   matrixc&               shur_s,    // matrix S in Schur form
   matrixc&               shur_t,    // matrix T in Schur form
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixc&                vsl,       // matrix of left Schur vectors VSL
   matrixc&                vsr        // matrix of right Schur vectors VSR
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixcf&               B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorcf&               alpha,     // vector of computed eigenvalues
   vectorcf&               beta,      // vector of eigenvalue divisors
   matrixcf&               shur_s,    // matrix S in Schur form
   matrixcf&               shur_t,    // matrix T in Schur form
   matrixcf&               vsl,       // matrix of left Schur vectors VSL
   matrixcf&               vsr        // matrix of right Schur vectors VSR
   );
```

**Parameters**

*B*

[in] The second matrix in the pair.

*jobvs*

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

*alpha*

[out] Vector of eigenvalues.

*beta*

[out] Vector of eigenvalue divisors.

*shur_s*

[out] Matrix S, block upper triangular Schur matrix (Schur form for the input matrix).

*shur_t*

[out] Matrix T, block upper triangular Schur matrix (Schur form for the second matrix in the pair).

*vsl*

[out] Matrix of left Schur vectors VSL.

*vsr*

[out] Matrix of right Schur vectors VSR.

**Return Value**

The function returns 'true' on success or 'false' if an error occurs.

**Note**

Computation depends on the *jobvs* parameter values.

© 2000-2025, MetaQuotes Ltd.

The second matrix in the pair must be the same size as the first (input) one.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

### EigenSolverShur

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVECTORS_N | Only eigenvalues are computed, without vectors. |
| EIGVECTORS_L | Only left eigenvectors are computed. |
| EIGVECTORS_R | Only right eigenvectors are computed. |
| EIGVECTORS_LR | Left and right eigenvectors are computed, eigenvalues are always computed. |

© 2000-2025, MetaQuotes Ltd.

# Symmetric Matrices

Functions for computing eigenvalues and eigenvectors of symmetric or Hermitian matrices using the divide and conquer algorithm, making the process efficient and fast. These methods can be applied to matrices of different data types, including real and complex numbers.

| Function | Action |
| --- | --- |
| EigenSymmetricDC | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD). |
| EigenSymmetricQR | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV). |
| EigenSymmetricRobust | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR). |
| EigenSymmetricBisect | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX). |
| EigenSymmetricDC2s | Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm (lapack functions SYEVD_2STAGE, HEEVD_2STAGE). |
| EigenSymmetricQR2s | Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal (lapack functions SYEV_2STAGE, HEEV_2STAGE). |
| EigenSymmetricRobust2s | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR_2STAGE, HEEVR_2STAGE). |
| EigenSymmetricBisect2s | Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| | to tridiagonal then using the bisection algorithm (lapack functions SYEVX_2STAGE, HEEVX_2STAGE). |
| EigenSymmetric2DC | Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. |
| EigenSymmetric2QR | Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. |
| EigenSymmetric2Bisect | Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricDC

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD).

### Computing for type matrix<double>

```
bool  matrix::EigenSymmetricDC(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricDC(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricDC(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricDC(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixcf&             eigen_vectors      // matrix of computed eigenvectors
   );
```

### Parameters

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricQR

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the QR algorithm (lapack functions SYEV, HEEV).

### Computing for type matrix<double>

```
bool  matrix::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixcf&             eigen_vectors      // matrix of computed eigenvectors
   );
```

### Parameters

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricRobust

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR).

### Computing for type matrix<double>

```
bool  matrix::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // Upper bound of the subset
   float                 abstol,            // absolute error tolerance
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf&              eigen_values,       // vector of computed eigenvectors
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

© 2000-2025, MetaQuotes Ltd.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricBisect

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX).

### Computing for type matrix<double>

```
bool  matrix::EigenSymmetricBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // Upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*V*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

© 2000-2025, MetaQuotes Ltd.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricDC2s

Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm (lapack functions SYEVD_2STAGE, HEEVD_2STAGE).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixcf&             eigen_vectors      // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. Not available in this release. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricQR2s

Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal (lapack functions SYEV_2STAGE, HEEV_2STAGE).

### Computing for type matrix<double>

```
bool  matrix::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixcf&             eigen_vectors      // matrix of computed eigenvectors
   );
```

### Parameters

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. Not available in this release. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricRobust2s

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR_2STAGE, HEEVR_2STAGE).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf&              eigen_values,       // vector of computed eigenvectors
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

© 2000-2025, MetaQuotes Ltd.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. Not available in this release. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetricBisect2s

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the bisection algorithm (lapack functions SYEVX_2STAGE, HEEVX_2STAGE).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorf&              eigen_values,       // vector of computed eigenvectors
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*V*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

© 2000-2025, MetaQuotes Ltd.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. Not available in this release. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetric2DC

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form

A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. If eigenvectors are desired, it uses a divide-and-conquer algorithm (lapack functions SYGVD, HEGVD).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrix&               B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors,      // matrix of computed eigenvectors
   matrix&               triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixf&              B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors,      // matrix of computed eigenvectors
   matrixf&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixc&              B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors,      // matrix of computed eigenvectors
   matrixc&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixcf&             B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors,      // matrix of computed eigenvectors
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixcf&            triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Parameters**

*itype*

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*B*

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

*triangular_factor*

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

| ID | Description |
|---|---|
| EIGS2TYPE_1 | 1: A*x = (lambda)*B*x |
| EIGS2TYPE_2 | 2: A*B*x = (lambda)*x |
| EIGS2TYPE_3 | 3: B*A*x = (lambda)*x |

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetric2QR

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form

A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. This method uses QR algorithm (lapack functions SYGV, HEGV).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrix&               B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors,      // matrix of computed eigenvectors
   matrix&               triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixf&              B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors,      // matrix of computed eigenvectors
   matrixf&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixc&              B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors,      // matrix of computed eigenvectors
   matrixc&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixcf&             B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors,      // matrix of computed eigenvectors
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixcf&             triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Parameters**

*itype*

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*B*

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

*triangular_factor*

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

| ID | Description |
|---|---|
| EIGS2TYPE_1 | 1: A*x = (lambda)*B*x |
| EIGS2TYPE_2 | 2: A*B*x = (lambda)*x |
| EIGS2TYPE_3 | 3: B*A*x = (lambda)*x |

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenSymmetric2Bisect

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form

A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues. This method uses bisection algorithm (lapack functions SYGVX, HEGVX).

**Computing for type matrix<double>**

```
bool  matrix::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   double                lower,              // lower bound of the subset
   double                upper,              // Upper bound of the subset
   double                abstol,             // absolute error tolerance
   matrix&               B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors,      // matrix of computed eigenvectors
   matrix&               triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   float                 lower,              // lower bound of the subset
   float                 upper,              // upper bound of the subset
   float                 abstol,             // absolute error tolerance
   matrixf&              B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors,      // matrix of computed eigenvectors
   matrixf&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   double                lower,              // lower bound of the subset
   double                upper,              // Upper bound of the subset
   double                abstol,             // absolute error tolerance
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixc&              B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors,      // matrix of computed eigenvectors
   matrixc&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   float                 lower,              // lower bound of the subset
   float                 upper,              // Upper bound of the subset
   float                 abstol,             // absolute error tolerance
   matrixcf&             B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors,      // matrix of computed eigenvectors
   matrixcf&             triangular_factor   // triangular factor from the Cholesky fa
   );
```

**Parameters**

*itype*

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

© 2000-2025, MetaQuotes Ltd.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*B*

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

*triangular_factor*

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

| ID | Description |
|---|---|
| EIGS2TYPE_1 | 1: A*x = (lambda)*B*x |
| EIGS2TYPE_2 | 2: A*B*x = (lambda)*x |
| EIGS2TYPE_3 | 3: B*A*x = (lambda)*x |

© 2000-2025, MetaQuotes Ltd.

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# Tridiagonal Matrices

Functions for computing eigenvalues and eigenvectors of symmetric tridiagonal matrices using various algorithms. Each function implements a specific solution method and supports matrix types double and float.

Common Parameters:

- jobv — Determines whether to compute eigenvectors (EIGVALUES_V) or only eigenvalues (EIGVALUES_N).
- range — Specifies the range of computed eigenvalues (BLASRANGE_A, BLASRANGE_V, BLASRANGE_I).
- lower and upper — Lower and upper bounds for computing a subset of the spectrum.
- abstol — Absolute error tolerance.

All functions operate on symmetric tridiagonal matrices and allow selecting the most suitable algorithm depending on performance and accuracy requirements.

| Function | Action |
|---|---|
| EigenTridiagonalDC | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEVD). |
| EigenTridiagonalQR | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEV). |
| EigenTridiagonalRobust | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack function STEVR). |
| EigenTridiagonalBisect | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the bisection algorithm (lapack function STEVX). |
| EigenTridiagonalQL | Compute all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm (lapack function STERF). |
| EigenTridiagonalDCQ | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEDC). |
| EigenTridiagonalQRQ | Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEQR). |
| EigenTridiagonalPosDefQ | Compute eigenvalues and eigenvectors of a symmetric positive definite (положительно |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| | определённая) tridiagonal matrix using the QR algorithm (lapack function PTEQR). |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalDC

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEVD).

**Computing for type matrix<double>**

```
bool  matrix::EigenTridiagonalDC(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenTridiagonalDC(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalQR

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEV).

**Computing for type matrix<double>**

```
bool  matrix::EigenTridiagonalQR(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenTridiagonalQR(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIG_VALUES**

An enumeration that specifies whether to calculate eigenvectors.

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are calculated, without vectors. |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalRobust

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack function STEVR).

**Computing for type matrix<double>**

```
bool  matrix::EigenTridiagonalRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenTridiagonalRobust(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

**Parameters**

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

© 2000-2025, MetaQuotes Ltd.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalBisect

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the bisection algorithm (lapack function STEVX).

### Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   double                lower,             // lower bound of the subset
   double                upper,             // Upper bound of the subset
   double                abstol,            // absolute error tolerance
   vector&               eigen_values,      // vector of computed eigenvectors
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalBisect(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,             // subset of eigenvalues to compute
   float                 lower,             // lower bound of the subset
   float                 upper,             // upper bound of the subset
   float                 abstol,            // absolute error tolerance
   vectorf&              eigen_values,      // vector of computed eigenvectors
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

### Parameters

*jobv*

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

*range*

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

*lower*

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*upper*

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

*abstol*

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

© 2000-2025, MetaQuotes Ltd.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

*eigen_values*

[out] Vector of eigenvalues.

*V*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIG_VALUES**

An enumeration defining the need to compute eigenvectors.

| ID | Description |
|---|---|
| EIGVALUES_V | Eigenvectors and eigenvalues are calculated. |
| EIGVALUES_N | Only eigenvalues are computed, without vectors. |

**ENUM_BLAS_RANGE**

An enumeration defining how right singular vectors should be computed.

| ID | Description |
|---|---|
| BLASRANGE_A | All singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| BLASRANGE_V | All singular or eigenvalues in the half-open interval (VL,VU] will be found. |
| BLASRANGE_I | The IL-th through IU-th singular or eigenvalues will be found. |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalQL

Compute all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm (lapack function STERF).

### Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalQL(
   vector&               eigen_values        // vector of computed eigenvalues
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalQL(
   vectorf&              eigen_values        // vector of computed eigenvalues
   );
```

### Parameters

*eigen_values*

[out] Vector of eigenvalues.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

The input must be a symmetric matrix in the tridiagonal form.

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalDCQ

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEDC). Unlike EigenTridiagonalDC, this method can be used to compute the eigenvectors of the original symmetric matrix. A symmetric matrix can be reduced to tridiagonal form using the ReduceSymmetricToTridiagonal method. The orthogonal matrix Q obtained from this transformation is then used to compute the eigenvectors of the original symmetric matrix.

**Computing for type matrix<double>**

```
bool  matrix::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrix&               Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixf&              Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixc&              Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors       // matrix of computed eigenvectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixcf&             Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

**Parameters**

*compv*

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

*Q*

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

© 2000-2025, MetaQuotes Ltd.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIGTRIDIAG_Z**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
|---|---|
| EIGCOMPZ_N | 'N': Compute eigenvalues only |
| EIGCOMPZ_V | 'V': Compute eigenvectors of original symmetric matrix also |
| EIGCOMPZ_I | 'I': Compute eigenvectors of tridiagonal matrix also |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalQRQ

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEQR). Unlike EigenTridiagonalQR, this method can be used to compute the eigenvectors of the original symmetric matrix. A symmetric matrix can be reduced to tridiagonal form using the ReduceSymmetricToTridiagonal method. The orthogonal matrix Q obtained from this transformation is then used to compute the eigenvectors of the original symmetric matrix.

### Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrix&               Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixf&              Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixc&              Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixcf&             Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

### Parameters

*compv*

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

*Q*

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

© 2000-2025, MetaQuotes Ltd.

*eigen_values*

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric matrix in the tridiagonal form.

**ENUM_EIGTRIDIAG_Z**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
|---|---|
| EIGCOMPZ_N | 'N': Compute eigenvalues only |
| EIGCOMPZ_V | 'V': Compute eigenvectors of original symmetric matrix also |
| EIGCOMPZ_I | 'I': Compute eigenvectors of tridiagonal matrix also |

© 2000-2025, MetaQuotes Ltd.

# EigenTridiagonalPosDefQ

Compute eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix using the QR algorithm (lapack function PTEQR). This method can be used to compute the eigenvectors of both the input tridiagonal matrix and the original positive definite symmetric matrix. This is achieved using the orthogonal matrix Q obtained during the reduction to tridiagonal form.

### Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrix&               Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixf&              Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complex>

```
bool  matrixc::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixc&              Q,                  // orthogonal matrix used in the reductio
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors       // matrix of computed eigenvectors
   );
```

### Computing for type matrix<complexf>

```
bool  matrixcf::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z     compv,              // compute eigenvectors or not
   matrixcf&             Q,                  // orthogonal matrix used in the reductio
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

### Parameters

*compv*

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

*Q*

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

*eigen_values*

© 2000-2025, MetaQuotes Ltd.

[out] Vector of eigenvalues.

*eigen_vectors*

[out] Matrix of eigenvectors.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric positive definite tridiagonal matrix.

**ENUM_EIGTRIDIAG_Z**

An enumeration that specifies whether to calculate eigenvectors.

| ID | Description |
|---|---|
| EIGCOMPZ_N | 'N': Compute eigenvalues only |
| EIGCOMPZ_V | 'V': Compute eigenvectors of original symmetric matrix also |
| EIGCOMPZ_I | 'I': Compute eigenvectors of tridiagonal matrix also |

© 2000-2025, MetaQuotes Ltd.

# Orthogonal Factorizations

OpenBLAS предоставляет ряд процедур для разложения общей прямоугольной матрицы A размера m на n в произведение ортогональной (в комплексном случае — унитарной) и треугольной (или, возможно, трапециевидной) матриц.

Реальная матрица Q называется ортогональной, если $Q^T Q = I$ ; комплексная матрица Q называется унитарной, если $Q^H Q = I$ . Ортогональные и унитарные матрицы обладают важным свойством — они сохраняют евклидову норму вектора:

$||x||_2 = ||Qx||_2$, если Q ортогональна или унитарна

Благодаря этому такие матрицы способствуют численной устойчивости, поскольку не усиливают ошибки округления.

Ортогональные разложения широко применяются при решении задач наименьших квадратов. Их также можно использовать для выполнения предварительных шагов при решении задач на собственные значения или сингулярные значения.

| Функция | Выполняемое действие |
|---|---|
| FactorizationQR | Вычисляет QR-разложение общей матрицы размера m на n: A = Q * R. LAPACK-функция GEQRF. |
| FactorizationQRNonNeg | Вычисляет QR-разложение общей матрицы размера m на n: A = Q * R, где R — верхнетреугольная матрица с неотрицательными элементами на диагонали. LATPACK-функция GEQRFP. |
| FactorizationQRPivot | Вычисляет QR-разложение общей матрицы размера m на n с перестановкой столбцов: A * P = Q * R. LAPACK-функция GEQP3. |
| FactorizationLQ | Computes the LQ factorization of a general m-by-n matrix: A = L * Q. Lapack function GELQF. |
| FactorizationQL | Computes the QL factorization of a general m-by-n matrix: A = Q * L. Lapack function GEQLF. |
| FactorizationRQ | Computes the RQ factorization of a general m-by-n matrix: A = R * Q. Lapack function GERQF. |

© 2000-2025, MetaQuotes Ltd.

# FactorizationQR

Computes the QR factorization of a general m-by-n matrix: A = Q * R. Lapack function GEQRF.

### Computing for type matrix<double>

```
bool  matrix::FactorizationQR(
   bool            reduced,      // calculation mode reduced or complete
   matrix&         Q,            // orthogonal matrix Q
   matrix&         R             // upper triangular matrix R
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationQR(
   bool            reduced,      // calculation mode reduced or complete
   matrixf&        Q,            // orthogonal matrix Q
   matrixf&        R             // upper triangular matrix R
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationQR(
   bool            reduced,      // calculation mode reduced or complete
   matrixc&        Q,            // unitary matrix Q
   matrixc&        R             // upper triangular matrix R
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationQR(
   bool            reduced,      // calculation mode reduced or complete
   matrixcf&       Q,            // unitary matrix Q
   matrixcf&       R             // upper triangular matrix R
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

*Q*

[out] Orthogonal or unitary matrix Q.

*R*

[out] Upper triangular matrix R.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# FactorizationQRNonNeg

Computes the QR factorization of a general m-by-n matrix: A = Q * R. R is an upper triangular matrix with nonnegative diagonal entries. Lapack function GEQRFP.

### Computing for type matrix<double>

```
bool  matrix::FactorizationQRNonNeg(
   bool             reduced,       // calculation mode reduced or complete
   matrix&          Q,             // orthogonal matrix Q
   matrix&          R              // upper triangular matrix R
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationQRNonNeg(
   bool             reduced,       // calculation mode reduced or complete
   matrixf&         Q,             // orthogonal matrix Q
   matrixf&         R              // upper triangular matrix R
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationQRNonNeg(
   bool             reduced,       // calculation mode reduced or complete
   matrixc&         Q,             // unitary matrix Q
   matrixc&         R              // upper triangular matrix R
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationQRNonNeg(
   bool             reduced,       // calculation mode reduced or complete
   matrixcf&        Q,             // unitary matrix Q
   matrixcf&        R              // upper triangular matrix R
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

*Q*

[out] Orthogonal or unitary matrix Q.

*R*

[out] Upper triangular matrix R.

### Return Value

© 2000-2025, MetaQuotes Ltd.

Return true if successful, otherwise false in case of an error.

**Note**

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# FactorizationQRPivot

Computes the QR factorization of a general m-by-n matrix with column pivoting: A * P = Q * R. Lapack function GEQP3.

### Computing for type matrix<double>

```
bool  matrix::FactorizationQRPivot(
   bool             reduced,       // calculation mode reduced or complete
   long[]&          jpvt,          // array with predefined permutations
   matrix&          Q,             // orthogonal matrix Q
   matrix&          R,             // upper triangular matrix R
   matrix&          P              // permutation matrix
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationQRPivot(
   bool             reduced,       // calculation mode reduced or complete
   long[]&          jpvt,          // array with predefined permutations
   matrixf&         Q,             // orthogonal matrix Q
   matrixf&         R,             // upper triangular matrix R
   matrixf&         P              // permutation matrix
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationQRPivot(
   bool             reduced,       // calculation mode reduced or complete
   long[]&          jpvt,          // array with predefined permutations
   matrixc&         Q,             // unitary matrix Q
   matrixc&         R,             // upper triangular matrix R
   matrixc&         P              // permutation matrix
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationQRPivot(
   bool             reduced,       // calculation mode reduced or complete
   long[]&          jpvt,          // array with predefined permutations
   matrixcf&        Q,             // unitary matrix Q
   matrixcf&        R,             // upper triangular matrix R
   matrixcf&        P              // permutation matrix
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

© 2000-2025, MetaQuotes Ltd.

*jpvt*

[in] Integer array of dimension n. if jpvt(i) ≠ 0, the i-th column of A is moved to the beginning of AP before the computation, and fixed in place during the computation. If jpvt(i) = 0, the i-th column of A is a free column (that is, it may be interchanged during the computation with any other free column). If array has zero size (or not initialized), then all the columns of A assumed to be free.

*Q*

[out] Orthogonal or unitary matrix Q.

*R*

[out] Upper triangular matrix R.

*P*

[out] Permutation matrix P of n-by-n sizes.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# FactorizationLQ

Computes the LQ factorization of a general m-by-n matrix: A = L * Q. Lapack function GELQF.

### Computing for type matrix<double>

```
bool  matrix::FactorizationLQ(
   bool            reduced,      // calculation mode reduced or complete
   matrix&         L,            // lower triangular matrix L
   matrix&         Q             // orthogonal matrix Q
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationLQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixf&        L,            // lower triangular matrix L
   matrixf&        Q             // orthogonal matrix Q
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationLQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixc&        L,            // lower triangular matrix L
   matrixc&        Q             // unitary matrix Q
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationLQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixcf&       L,            // lower triangular matrix L
   matrixcf&       Q             // unitary matrix Q
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices L, Q calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices L, Q with dimensions (M,N), (N,N).

*L*

[out] Lower triangular matrix L.

*Q*

[out] Orthogonal or unitary matrix Q.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

If *reduced* is true

If m <= n, matrix L is of m-by-m sizes, matrix Q is of m-by-n sizes.

If m > n, matrix L is of m-by-n sizes, matrix Q is of n-by-n sizes.

If *reduced* is false, matrix L is of m-by-n sizes, matrix Q is of n-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# FactorizationQL

Computes the QL factorization of a general m-by-n matrix: A = Q * L. Lapack function GEQLF.

### Computing for type matrix<double>

```
bool  matrix::FactorizationQL(
   bool            reduced,      // calculation mode reduced or complete
   matrix&         Q,            // orthogonal matrix Q
   matrix&         L             // lower triangular matrix L
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationQL(
   bool            reduced,      // calculation mode reduced or complete
   matrixf&        Q,            // orthogonal matrix Q
   matrixf&        L             // lower triangular matrix L
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationQL(
   bool            reduced,      // calculation mode reduced or complete
   matrixc&        Q,            // unitary matrix Q
   matrixc&        L             // lower triangular matrix L
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationQL(
   bool            reduced,      // calculation mode reduced or complete
   matrixcf&       Q,            // unitary matrix Q
   matrixcf&       L             // lower triangular matrix L
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices Q, L calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, L with dimensions (M,M), (M,N).

*Q*

[out] Orthogonal or unitary matrix Q.

*L*

[out] Lower triangular matrix L.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix L is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix L is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix L is of m-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# FactorizationRQ

Computes the RQ factorization of a general m-by-n matrix: A = R * Q. Lapack function GERQF.

### Computing for type matrix<double>

```
bool  matrix::FactorizationRQ(
   bool            reduced,      // calculation mode reduced or complete
   matrix&         R,            // upper triangular matrix R
   matrix&         Q             // orthogonal matrix Q
   );
```

### Computing for type matrix<float>

```
bool  matrix::FactorizationRQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixf&        R,            // upper triangular matrix R
   matrixf&        Q             // orthogonal matrix Q
   );
```

### Computing for type matrix<complex>

```
bool  matrix::FactorizationRQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixc&        R,            // upper triangular matrix R
   matrixc&        Q             // unitary matrix Q
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::FactorizationRQ(
   bool            reduced,      // calculation mode reduced or complete
   matrixcf&       R,            // upper triangular matrix R
   matrixcf&       Q             // unitary matrix Q
   );
```

### Parameters

*reduced*

[in] Calculation mode. If reduced is true then matrices R, Q calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices L, Q with dimensions (M,N), (N,N).

*R*

[out] Upper triangular matrix R.

*Q*

[out] Orthogonal or unitary matrix Q.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

If *reduced* is true

If m <= n, matrix R is of m-by-m sizes, matrix Q is of m-by-n sizes.

If m > n, matrix R is of m-by-n sizes, matrix Q is of n-by-n sizes.

If *reduced* is false, matrix R is of m-by-n sizes, matrix Q is of n-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# Linear Equations

This section provides functions for solving systems of linear equations of the form A·X = B, where A is a square coefficient matrix, B is the right-hand side, and X is the solution matrix or vector. The methods support various data types (double, float, complex, complexf) and are based on LAPACK functions to ensure high precision and performance.

The functions are categorized based on the type of the coefficient matrix A. Each function is overloaded to support both matrix and vector forms for the right-hand side (B) and the solution (X).

| Function | Action |
| --- | --- |
| LinearEquationsSolution | Computes the solution to the system of linear equations with a square coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GESV. |
| LinearEquationsSolutionTriangular | Computes the solution to the system of linear equations with a square triangular coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function TRTRS. |
| LinearEquationsSolutionSy | Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack functions SYSV, HESV. |
| LinearEquationsSolutionComplexSy | Computes the solution to the system of linear equations with a complex symmetric (not Hermitian conjugated!) matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n complex symmetric matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function SYSV. |
| LinearEquationsSolutionSyPD | Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated positive-definite matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated positive-definite matrix, the columns of matrix B are individual right-hand sides, |

© 2000-2025, MetaQuotes Ltd.

| Function | Action |
| --- | --- |
| | and the columns of X are the corresponding solutions. Lapack function POSV. |
| LinearEquationsSolutionGeTrid | Computes the solution to the system of linear equations with a general (nonsymmetric) tridiagonal coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GTSV. |
| LinearEquationsSolutionSyTridPD | Computes the solution to the system of linear equations with a symmetric tridiagonal positive-definite coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function PTSV. |

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolution

Computes the solution to the system of linear equations with a square coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GESV.

### Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolution(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

### Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolution(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

### Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolution(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolution(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&       B,              // right hand side vector B
   vectorcf&       X               // solution vector X
   );
```

### Parameters

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionTriangular

Computes the solution to the system of linear equations with a square triangular coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function TRTRS.

**Computing for type matrix<double>**

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   matrix&             B,            // right hand side matrix B
   matrix&             X             // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   vector&             B,            // right hand side vector B
   vector&             X             // solution vector X
   );
```

**Computing for type matrix<float>**

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   matrixf&            B,            // right hand side matrix B
   matrixf&            X             // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   vectorf&            B,            // right hand side vector B
   vectorf&            X             // solution vector X
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   matrixc&            B,            // right hand side matrix B
   matrixc&            X             // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   vectorc&            B,            // right hand side vector B
   vectorc&            X             // solution vector X
   );
```

**Computing for type matrix<complexf>**

© 2000-2025, MetaQuotes Ltd.

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   matrixcf&           B,            // right hand side matrix B
   matrixcf&           X             // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,        // form of the system of equations
   vectorcf&           B,            // right hand side vector B
   vectorcf&           X             // solution vector X
   );
```

**Parameters**

*trans*

[in] ENUM_EQUATIONS_FORM enumeration value which specifies the form of the system of equations.

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

**ENUM_EQUATIONS_FORM**

An enumeration defining which form of the equations' system calculated.

| ID | Description |
|---|---|
| EQUATIONSFORM_N | 'N': A * X = B (No transpose) |
| EQUATIONSFORM_T | 'T': A**T * X = B (Transpose) |
| EQUATIONSFORM_C | 'C': A**H * X = B (Conjugate transpose) |

In case of real matrices the value EQUATIONSFORM_C assumed as Transpose.

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionSy

Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or unitary matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack functions SYSV, HESV.

**Computing for type matrix<double>**

```
bool  matrix::LinearEquationsSolutionSy(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

**Computing for type matrix<float>**

```
bool  matrix::LinearEquationsSolutionSy(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::LinearEquationsSolutionSy(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::LinearEquationsSolutionSy(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&       B,              // right hand side vector B
   vectorcf&       X               // solution vector X
   );
```

### Parameters

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a symmetric (Hermitian conjugated), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionComplexSy

Computes the solution to the system of linear equations with a complex symmetric (not Hermitian conjugated!) matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n complex symmetric matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function SYSV.

**Computing for type matrix<complex>**

```
bool  matrix::LinearEquationsSolutionComplexSy(
   matrixc&        B,             // right hand side matrix B
   matrixc&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionComplexSy(
   vectorc&        B,             // right hand side vector B
   vectorc&        X              // solution vector X
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::LinearEquationsSolutionComplexSy(
   matrixcf&       B,             // right hand side matrix B
   matrixcf&       X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionComplexSy(
   vectorcf&       B,             // right hand side vector B
   vectorcf&       X              // solution vector X
   );
```

**Parameters**

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a complex symmetric, upper triangular or lower triangular matrix. Triangular matrices are assumed to be complex symmetric.

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionSyPD

Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated positive-definite matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or unitary positive-definite matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function POSV.

**Computing for type matrix<double>**

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

**Computing for type matrix<float>**

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&       B,              // right hand side vector B
   vectorcf&       X               // solution vector X
   );
```

### Parameters

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a symmetric (Hermitian conjugated), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionGeTrid

Computes the solution to the system of linear equations with a general (nonsymmetric) tridiagonal coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GTSV.

**Computing for type matrix<double>**

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

**Computing for type matrix<float>**

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&       B,              // right hand side vector B
   vectorcf&       X               // solution vector X
   );
```

### Parameters

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

© 2000-2025, MetaQuotes Ltd.

# LinearEquationsSolutionSyTridPD

Computes the solution to the system of linear equations with a symmetric tridiagonal positive-definite coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function PTSV.

### Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

### Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

### Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&       B,              // right hand side vector B
   vectorcf&       X               // solution vector X
   );
```

### Parameters

*B*

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

*X*

[out] Matrix or vector X with solutions of linear equations system.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

© 2000-2025, MetaQuotes Ltd.

# Factorizations

The Factorizations section contains functions for performing various types of matrix factorizations used in numerical solutions of linear systems, stability analysis, and other linear algebra tasks. These factorizations transform the original matrix into simpler forms, making subsequent computations more efficient. All functions are implemented using LAPACK routines and support the types double, float, complex, and complexf.

The functions in this section are used for:

- Preprocessing matrices when solving systems of linear equations;
- Computing determinants, ranks, and matrix inverses;
- Assessing the stability of numerical methods;
- Solving problems in spectral theory and optimization methods.

Matrix factorization is a critical step in many linear algebra algorithms, and this section provides access to the most efficient and well-established factorization techniques.

| Function | Action |
| --- | --- |
| FactorizationPLU | Computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges. The factorization has the form A = P * L * U. Lapack function GETRF. |
| FactorizationPLUGeTrid | Computes an LU factorization of a general (non-symmetric) tridiagonal N-by-N matrix A using elimination with partial pivoting and row interchanges. The factorization has the form A = P * L * U. Lapack function GTTRF. |
| FactorizationLDL | Computes the factorization of a real symmetric or complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. Lapack functions SYTRF, HETRF. |
| FactorizationLDLSyTridPD | Forms the factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite tridiagonal matrix A. Lapack function PTTRF. |
| FactorizationCholesky | Computes the factorization of a real symmetric or complex Hermitian positive-definite matrix A. Lapack function POTRF. |
| FactorizationCholeskySyPS | Computes the Cholesky factorization with complete pivoting of a real symmetric (complex Hermitian) positive semidefinite N-by-N matrix. Lapack function PSTRF. |

© 2000-2025, MetaQuotes Ltd.

# FactorizationPLU

Computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges. The factorization has the form

A = P * L * U

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n). Lapack function GETRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationPLU(
   matrix&          P,             // permutation matrix P
   matrix&          L,             // lower triangular matrix L
   matrix&          U              // upper triangular matrix U
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationPLU(
   matrixf&         P,             // permutation matrix P
   matrixf&         L,             // lower triangular matrix L
   matrixf&         U              // upper triangular matrix U
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationPLU(
   matrixc&         P,             // permutation matrix P
   matrixc&         L,             // lower triangular matrix L
   matrixc&         U              // upper triangular matrix U
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationPLU(
   matrixcf&        P,             // permutation matrix P
   matrixcf&        L,             // lower triangular matrix L
   matrixcf&        U              // upper triangular matrix U
   );
```

**Parameters**

*P*

[out] Permutation matrix P.

*L*

[out] Lower triangular matrix L with unit diagonal elements.

*U*

[out] Upper triangular matrix U.

© 2000-2025, MetaQuotes Ltd.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

# FactorizationPLUGeTrid

Computes an LU factorization of a general (non-symmetric) tridiagonal N-by-N matrix A using elimination with partial pivoting and row interchanges. The factorization has the form

A = P * L * U

where P is a permutation matrix, L is lower triangular with unit diagonal elements, and U is upper triangular. Lapack function GTTRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationPLUGeTrid(
   matrix&          P,             // permutation matrix P
   matrix&          L,             // lower triangular matrix L
   matrix&          U              // upper triangular matrix U
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationPLUGeTrid(
   matrixf&         P,             // permutation matrix P
   matrixf&         L,             // lower triangular matrix L
   matrixf&         U              // upper triangular matrix U
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationPLUGeTrid(
   matrixc&         P,             // permutation matrix P
   matrixc&         L,             // lower triangular matrix L
   matrixc&         U              // upper triangular matrix U
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationPLUGeTrid(
   matrixcf&        P,             // permutation matrix P
   matrixcf&        L,             // lower triangular matrix L
   matrixcf&        U              // upper triangular matrix U
   );
```

**Parameters**

*P*

[out] Permutation matrix P.

*L*

[out] Lower triangular matrix L with unit diagonal elements.

*U*

[out] Upper triangular matrix U.

© 2000-2025, MetaQuotes Ltd.

### Return Value

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

# FactorizationLDL

Computes the factorization of a real symmetric or complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is:

A = L * D * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * D * U in case of upper triangular matrix A

where L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements. D is a symmetric block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. Lapack functions SYTRF, HETRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationLDL(
   matrix&          L,             // lower or upper triangular matrix
   matrix&          D              // diagonal matrix D
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationLDL(
   matrixf&         L,             // lower or upper triangular matrix
   matrixf&         D              // diagonal matrix D
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationLDL(
   matrixc&         L,             // lower or upper triangular matrix
   matrixc&         D              // diagonal matrix D
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationLDL(
   matrixcf&        L,             // lower or upper triangular matrix
   matrixcf&        D              // diagonal matrix D
   );
```

**Parameters**

*L*

[out] Lower or upper triangular matrix with unit diagonal elements.

*D*

[out] Symmetric block-diagonal matrix D.

**Return Value**

© 2000-2025, MetaQuotes Ltd.

Return true if successful, otherwise false in case of an error.

**Note**

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# FactorizationLDLSyTridPD

Forms the factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite tridiagonal matrix A:

A = L * D * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * D * U in case of upper triangular matrix A

where L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements. D is diagonal matrix. Lapack function PTTRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationLDLSyTridPD(
   matrix&         L,             // lower or upper triangular matrix
   matrix&         D              // diagonal matrix D
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixf&        L,             // lower or upper triangular matrix
   matrixf&        D              // diagonal matrix D
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixc&        L,             // lower or upper triangular matrix
   matrixc&        D              // diagonal matrix D
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixcf&       L,             // lower or upper triangular matrix
   matrixcf&       D              // diagonal matrix D
   );
```

**Parameters**

*L*

[out] Lower or upper triangular matrix with unit diagonal elements.

*D*

[out] Diagonal matrix D.

**Return Value**

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

**Note**

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# FactorizationCholesky

Computes the factorization of a real symmetric or complex Hermitian positive-definite matrix A. The factorization has the form:

A = L * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * U in case of upper triangular matrix A

where L is lower triangular, U is upper triangular. Lapack function POTRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationCholesky(
   matrix&          L              // lower or upper triangular matrix
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationCholesky(
   matrixf&         L              // lower or upper triangular matrix
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationCholesky(
   matrixc&         L              // lower or upper triangular matrix
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationCholesky(
   matrixcf&        L              // lower or upper triangular matrix
   );
```

**Parameters**

*L*

[out] Lower or upper triangular matrix.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# FactorizationCholeskySyPS

Computes the Cholesky factorization with complete pivoting of a real symmetric (complex Hermitian) positive semidefinite N-by-N matrix. The form of the factorization is:

P**T * A * P = L * L**T in case of lower triangular or symmetric matrix A

or

P**T * A * P = U**T * U in case of upper triangular matrix A

where P is a permutation matrix, L is lower triangular, U is upper triangular. Lapack function PSTRF.

**Computing for type matrix<double>**

```
bool  matrix::FactorizationCholeskySyPS(
   double          tol           // tolerance
   matrix&         P,            // permutation matrix P
   matrix&         L             // lower or upper triangular matrix
   );
```

**Computing for type matrix<float>**

```
bool  matrix::FactorizationCholeskySyPS(
   float           tol           // tolerance
   matrixf&        P,            // permutation matrix P
   matrixf&        L             // lower or upper triangular matrix
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::FactorizationCholeskySyPS(
   double          tol           // tolerance
   matrixc&        P,            // permutation matrix P
   matrixc&        L             // lower or upper triangular matrix
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::FactorizationCholeskySyPS(
   float           tol           // tolerance
   matrixcf&       P,            // permutation matrix P
   matrixcf&       L             // lower or upper triangular matrix
   );
```

**Parameters**

*tol*

[in] User defined tolerance. If tol < 0, then n*ε*max(A[k,k]), where ε is the machine precision, will be used. The algorithm terminates at the (k-1)st step, if the pivot <=tol.

*P*

[out] Permutation matrix P.

© 2000-2025, MetaQuotes Ltd.

*L*

[out] Lower or upper triangular matrix.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

© 2000-2025, MetaQuotes Ltd.

# Dynamic Mode Decomposition

The Dynamic Mode Decomposition (DMD) functions are designed to compute an evolution operator from data snapshots, enabling efficient analysis of linear dynamics in extended phase spaces. The methods provide memory-efficient representations of evolution operators, with DynamicModeDecompositionQR incorporating QR-based data compression for improved computational efficiency.

| Function | Action |
| --- | --- |
| DynamicModeDecomposition | Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices. Lapack function GEDMD. |
| DynamicModeDecompositionQR | Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices, using a QR factorization based compression of the data. Lapack function GEDMDQ. |

The Dynamic mode decomposition (DMD) method is an algorithm for searching for an evolution operator (inverse operator problem solutions) in a finite-dimensional problem solution space (numerical or experimentally obtained) in a set of solutions (slices, "snapshots") in some consecutive moments of time. Expansion of the phase space due to the use of a nonlinear basis (relative to the variables of the problem) allows us to construct a global linear operator describing a linear evolution in the extended "rectifying space" (the Coopman operator) and the Perron-Frobenius operator that is its adjoint one. The DMD method is equivalent to a compressed representation of a linear evolution operator in the form of a product of rectangular matrices, which provides significant savings in the required memory during calculations.

© 2000-2025, MetaQuotes Ltd.

# DynamicModeDecomposition

Compute the Dynamic Mode Decomposition(по-русски "разложение по динамическим модам") (DMD) for a pair of data snapshot matrices. Lapack function GEDMD. For the input matrices X and Y such that Y = A*X with an unaccessible matrix A, GEDMD computes a certain number of Ritz pairs of A using the standard Rayleigh-Ritz extraction from a subspace of range(X) that is determined using the leading left singular vectors of X. Optionally, GEDMD returns the residuals of the computed Ritz pairs, the information needed for a refinement of the Ritz vectors, or the eigenvectors of the Exact DMD. Both matrices must be the same size.

### Computing for type matrix<double>

```
bool  matrix::DynamicModeDecomposition(
   matrix&               B,                  // second snapshot matrix in the pai
   ENUM_DMD_SCALE        jobs,               // determines whether the initial da
   ENUM_DMD_EIGV         jobz,               // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,               // determines whether to compute the
   ENUM_DMD_REFINE       jobf,               // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,             // allows for a selection of the SVD
   long                  nrnk,               // determines the mode how to comput
   double                tol,                // the tolerance for truncating smal
   vectorc&              eigen_values,       // vector of eigenvalues (Ritz value
   matrix&               left_vectors,       // matrix of left singular vectors
   matrix&               Z                   // matrix of Ritz vectors
   vector&               residuals,          // residuals for the Ritz pairs
   matrix&               res_vectors,        // residual vectors for the Ritz pai
   matrix&               B,                  // computed vectors
   matrix&               W,                  // eigenvectors of the matrix Rayle
   matrix&               S                   // computed vectors
   );
```

### Computing for type matrix<float>

```
bool  matrixf::DynamicModeDecomposition(
   matrixf&              B,                  // second snapshot matrix in the pai
   ENUM_DMD_SCALE        jobs,               // determines whether the initial da
   ENUM_DMD_EIGV         jobz,               // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,               // determines whether to compute the
   ENUM_DMD_REFINE       jobf,               // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,             // allows for a selection of the SVD
   long                  nrnk,               // determines the mode how to comput
   float                 tol,                // the tolerance for truncating smal
   vectorcf&             eigen_values,       // vector of eigenvalues (Ritz value
   matrixf&              left_vectors,       // matrix of left singular vectors
   matrixf&              Z                   // matrix of Ritz vectors
   vectorf&              residuals,          // residuals for the Ritz pairs
   matrixf&              res_vectors,        // residual vectors for the Ritz pai
   matrixf&              B,                  // computed vectors
   matrixf&              W,                  // eigenvectors of the matrix Rayle
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixf&              S                       // computed vectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::DynamicModeDecomposition(
   matrixc&              B,                      // second snapshot matrix in the pa
   ENUM_DMD_SCALE        jobs,                   // determines whether the initial da
   ENUM_DMD_EIGV         jobz,                   // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                   // determines whether to compute the
   ENUM_DMD_REFINE       jobf,                   // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,                 // allows for a selection of the SVD
   long                  nrnk,                   // determines the mode how to comput
   double                tol,                    // the tolerance for truncating smal
   vectorc&              eigen_values,           // vector of eigenvalues (Ritz value
   matrixc&              left_vectors,           // matrix of left singular vectors
   matrixc&              Z                       // matrix of Ritz vectors
   vector&               residuals,              // residuals for the Ritz pairs
   matrixc&              res_vectors,            // residual vectors for the Ritz pa
   matrixc&              B,                      // computed vectors
   matrixc&              W,                      // eigenvectors of the matrix Rayle
   matrixc&              S                       // computed vectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::DynamicModeDecomposition(
   matrixcf&             B,                      // second snapshot matrix in the pa
   ENUM_DMD_SCALE        jobs,                   // determines whether the initial da
   ENUM_DMD_EIGV         jobz,                   // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                   // determines whether to compute the
   ENUM_DMD_REFINE       jobf,                   // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,                 // allows for a selection of the SVD
   long                  nrnk,                   // determines the mode how to comput
   float                 tol,                    // the tolerance for truncating smal
   vectorcf&             eigen_values,           // vector of eigenvalues (Ritz value
   matrixcf&             left_vectors,           // matrix of left singular vectors
   matrixcf&             Z                       // matrix of Ritz vectors
   vectorf&              residuals,              // residuals for the Ritz pairs
   matrixcf&             res_vectors,            // residual vectors for the Ritz pa
   matrixcf&             B,                      // computed vectors
   matrixcf&             W,                      // eigenvectors of the matrix Rayle
   matrixcf&             S                       // computed vectors
   );
```

**Parameters**

*Y*

[in] The second snapshot matrix in the pair.

© 2000-2025, MetaQuotes Ltd.

*jobs*

[in] Value from the ENUM_DMD_SCALE enumeration which determines whether the initial data snapshots are scaled by a diagonal matrix.

*jobz*

[in] Value from the ENUM_DMD_EIGV enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

*jobr*

[in] Value from the ENUM_DMD_RESIDUALS enumeration which determines whether to compute the residuals.

*jobf*

[in] Value from the ENUM_DMD_REFINE enumeration which specifies whether to store information needed for post-processing (e.g. computing refined Ritz vectors).

*whtsvd*

[in] Value from the ENUM_SVD_ALG enumeration which allows for a selection of the SVD algorithm from the LAPACK library.

*nrnk*

[in] Value determines the mode how to compute the numerical rank, i.e. how to truncate small singular values of the input matrix X. If

NRNK = -1 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(1). This option is recommended.

NRNK = -2 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(i-1). This option is included for R&D purposes. It requires highly accurate SVD, which may not be feasible.

The numerical rank can be enforced by using positive value of NRNK as follows: 0 < NRNK <= N :: at most NRNK largest singular values will be used. If the number of the computed nonzero singular values is less than NRNK, then only those nonzero values will be used and the actually used dimension is less than NRNK.

*tol*

[in] The tolerance for truncating small singular values. 0 <= TOL < 1

*eigen_values*

[out] Vector of eigenvalues of size K. The leading K (K<=N) entries of EIGS contain the computed eigenvalues (Ritz values).

*left_vectors*

[out] Matrix of left singular vectors. The leading K columns contain a POD basis (POD - Proper Orthogonal Decomposition, правильная ортогональная декомпозиция), i.e. the leading K left singular vectors of the input data matrix X All N columns of contain all left singular vectors of the input matrix X.

*Z*

[out] Matrix of Ritz vectors. If JOBZ =='V' then Z contains the Ritz vectors. Z(:,i) is an eigenvector of the i-th Ritz value; ||Z(:,i)||_2=1. If JOBZ == 'F', then the Z(:,i)'s are given implicitly as the columns of X(:,1:K)*W(1:K,1:K), i.e. X(:,1:K)*W(:,i) is an eigenvector corresponding to EIGS(i). The columns of W(1:k,1:K) are the computed eigenvectors of the K-by-K Rayleigh quotient.

© 2000-2025, MetaQuotes Ltd.

*residuals*

[out] Residuals for the K computed Ritz pairs.

*res_vectors*

[out] Residual vectors for the K computed Ritz pairs.

*B*

[out] M-by-K matrix (K<=N).

If JOBF =='R', B(1:M,1:K) contains A*U(:,1:K), and can be used for computing the refined vectors; see further details in the provided references.

If JOBF == 'E', B(1:M,1:K) contains A*U(:,1:K)*W(1:K,1:K), which are the vectors from the Exact DMD, up to scaling by the inverse eigenvalues.

If JOBF =='N', then B is not referenced.

*W*

[out] K-by-K matrix (K<=N). Contains the K computed eigenvectors of the matrix Rayleigh quotient. The Ritz vectors (returned in Z) are the product of X (containing a POD basis for the input matrix X) and W.

*S*

[out] K-by-K matrix (K<=N). The array S(1:K,1:K) is used for the matrix Rayleigh quotient. This content is overwritten during the eigenvalue decomposition by GEEV.

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The number of matrices rows (M) must not be less than the number of columns (N).

### ENUM_DMD_SCALE

An enumeration that determines whether the initial data snapshots are scaled by a diagonal matrix.

| ID | Description |
|---|---|
| DMDSCALE_S | 'S': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that X*D has unit nonzero columns |
| DMDSCALE_C | 'C': The snapshots are scaled as with the 'S' option.If it is found that an i-th column of X is zero vector and the corresponding i-th column of Y is non-zero, then the i-th column of Y is set to zero |
| DMDSCALE_Y | 'Y': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that Y*D has unit nonzero columns |
| DMDSCALE_N | 'N': No data scaling |

© 2000-2025, MetaQuotes Ltd.

**ENUM_DMD_EIGV**

An enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

| ID | Description |
|---|---|
| DMDEIGV_V | 'V': The eigenvectors (Koopman modes) will be computed |
| DMDEIGV_F | 'F': The eigenvectors (Koopman modes) will be returned in factored form |
| DMDEIGV_N | 'N': The eigenvectors are not computed |

**ENUM_DMD_RESIDUALS**

An enumeration which determines whether to compute the residuals.

| ID | Description |
|---|---|
| DMDRESIDUALS_R | 'R': The residuals for the computed eigenpairs will be computed |
| DMDRESIDUALS_N | 'N': The residuals are not computed |

**ENUM_DMD_REFINE**

An enumeration which specifies whether to store information needed for post-processing.

| ID | Description |
|---|---|
| DMDREFINE_R | 'R': The matrix needed for the refinement of the Ritz vectors is computed and stored in the array B |
| DMDREFINE_E | 'E': The unscaled eigenvectors of the Exact DMD are computed and returned in the array B |
| DMDREFINE_N | 'N': No eigenvector refinement data is computed |

**ENUM_SVD_ALG**

An enumeration selecting the SVD algorythm.

| ID | Description |
|---|---|
| SVDALG_1 | 1: GESVD (the QR SVD algorithm) |
| SVDALG_2 | 2: GESDD (the Divide and Conquer algorithm) |

© 2000-2025, MetaQuotes Ltd.

| ID | Description |
| --- | --- |
| SVDALG_3 | 3: GESVDQ (the preconditioned QR SVD; this and 4 are the most accurate options) |
| SVDALG_4 | 4: GEJSV (the preconditioned Jacobi SVD; this and 3 are the most accurate options) |

© 2000-2025, MetaQuotes Ltd.

# DynamicModeDecompositionQR

Compute the Dynamic Mode Decomposition(по-русски "декомпозиция динамических мод") (DMD) for a pair of data snapshot matrices, using a QR factorization based compression of the data. Lapack function GEDMDQ. For the input matrices X and Y such that Y = A*X with an unaccessible matrix A, GEDMDQ computes a certain number of Ritz pairs of A using the standard Rayleigh-Ritz extraction from a subspace of range(X) that is determined using the leading left singular vectors of X. Optionally, GEDMDQ returns the residuals of the computed Ritz pairs, the information needed for a refinement of the Ritz vectors, or the eigenvectors of the Exact DMD.

The input M-by-N matrix F. The columns of F are the sequence of data snapshots from a single trajectory, taken at equidistant discrete times. It is assumed that the column norms of F are in the range of the normalized floating point numbers.

**Computing for type matrix<double>**

```
bool  matrix::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                 // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                 // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                 // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                 // specifies whether to explicitly c
   ENUM_DMDQ_R           jobt,                 // specifies whether to return the u
   ENUM_DMD_REFINE       jobf,                 // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,               // allows for a selection of the SVD
   long                  nrnk,                 // determines the mode how to comput
   double                tol,                  // the tolerance for truncating smal
   vectorc&              eigen_values,         // vector of eigenvalues (Ritz value
   matrix&               Q,                    // Q factor from the QR factorizatio
   matrix&               R,                    // R factor from the QR factorizatio
   matrix&               left_vectors,         // matrix of left singular vectors
   matrix&               Z                     // matrix of Ritz vectors
   vector&               residuals,            // residuals for the Ritz pairs
   matrix&               B,                    // computed vectors
   matrix&               V,                    // eigenvectors of the matrix Rayle
   matrix&               S                     // computed vectors
   );
```

**Computing for type matrix<float>**

```
bool  matrixf::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                 // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                 // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                 // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                 // specifies whether to explicitly c
   ENUM_DMDQ_R           jobt,                 // specifies whether to return the u
   ENUM_DMD_REFINE       jobf,                 // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,               // allows for a selection of the SVD
   long                  nrnk,                 // determines the mode how to comput
   float                 tol,                  // the tolerance for truncating smal
```

© 2000-2025, MetaQuotes Ltd.

```
   vectorcf&            eigen_values,          // vector of eigenvalues (Ritz value
   matrixf&             Q,                     // Q factor from the QR factorizatio
   matrixf&             R,                     // R factor from the QR factorizatio
   matrixf&             left_vectors,          // matrix of left singular vectors
   matrixf&             Z                      // matrix of Ritz vectors
   vectorf&             residuals,             // residuals for the Ritz pairs
   matrixf&             B,                     // computed vectors
   matrixf&             V,                     // eigenvectors of the matrix Rayle
   matrixf&             S                      // computed vectors
   );
```

**Computing for type matrix<complex>**

```
bool  matrixc::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE       jobs,                  // determines whether the initial da
   ENUM_DMDQ_EIGV       jobz,                  // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS   jobr,                  // determines whether to compute the
   ENUM_DMDQ_Q          jobq,                  // specifies whether to explicitly c
   ENUM_DMDQ_R          jobt,                  // specifies whether to return the u
   ENUM_DMD_REFINE      jobf,                  // specifies whether to store inform
   ENUM_SVD_ALG         whtsvd,                // allows for a selection of the SVD
   long                 nrnk,                  // determines the mode how to comput
   double               tol,                   // the tolerance for truncating smal
   vectorc&             eigen_values,          // vector of eigenvalues (Ritz value
   matrixc&             Q,                     // Q factor from the QR factorizatio
   matrixc&             R,                     // R factor from the QR factorizatio
   matrixc&             left_vectors,          // matrix of left singular vectors
   matrixc&             Z                      // matrix of Ritz vectors
   vector&              residuals,             // residuals for the Ritz pairs
   matrixc&             B,                     // computed vectors
   matrixc&             V,                     // eigenvectors of the matrix Rayle
   matrixc&             S                      // computed vectors
   );
```

**Computing for type matrix<complexf>**

```
bool  matrixcf::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE       jobs,                  // determines whether the initial da
   ENUM_DMDQ_EIGV       jobz,                  // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS   jobr,                  // determines whether to compute the
   ENUM_DMDQ_Q          jobq,                  // specifies whether to explicitly c
   ENUM_DMDQ_R          jobt,                  // specifies whether to return the u
   ENUM_DMD_REFINE      jobf,                  // specifies whether to store inform
   ENUM_SVD_ALG         whtsvd,                // allows for a selection of the SVD
   long                 nrnk,                  // determines the mode how to comput
   float                tol,                   // the tolerance for truncating smal
   vectorcf&            eigen_values,          // vector of eigenvalues (Ritz value
   matrixcf&            Q,                     // Q factor from the QR factorizatio
```

© 2000-2025, MetaQuotes Ltd.

```
   matrixcf&            R,                  // R factor from the QR factorizatio
   matrixcf&            left_vectors,       // matrix of left singular vectors
   matrixcf&            Z                   // matrix of Ritz vectors
   vectorf&             residuals,          // residuals for the Ritz pairs
   matrixcf&            B,                  // computed vectors
   matrixcf&            V,                  // eigenvectors of the matrix Rayle
   matrixcf&            S                   // computed vectors
   );
```

**Parameters**

*jobs*

[in] Value from the ENUM_DMD_SCALE enumeration which determines whether the initial data snapshots are scaled by a diagonal matrix. The data snapshots are the columns of F. The leading N-1 columns of F are denoted X and the trailing N-1 columns are denoted Y.

*jobz*

[in] Value from the ENUM_DMDQ_EIGV enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

*jobr*

[in] Value from the ENUM_DMD_RESIDUALS enumeration which determines whether to compute the residuals.

*jobq*

[in] Value from the ENUM_DMDQ_Q enumeration which specifies whether to explicitly compute and return the unitary matrix from the QR factorization of the data snapshot matrix.

*jobt*

[in] Value from the ENUM_DMDQ_R enumeration which specifies whether to return the upper triangular factor from the QR factorization of the data snapshot matrix.

*jobf*

[in] Value from the ENUM_DMD_REFINE enumeration which specifies whether to store information needed for post-processing (e.g. computing refined Ritz vectors).

*whtsvd*

[in] Value from the ENUM_SVD_ALG enumeration which allows for a selection of the SVD algorithm from the LAPACK library.

*nrnk*

[in] Value determines the mode how to compute the numerical rank, i.e. how to truncate small singular values of the input matrix X. If

NRNK = -1 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(1). This option is recommended.

NRNK = -2 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(i-1). This option is included for R&D purposes. It requires highly accurate SVD, which may not be feasible.

The numerical rank can be enforced by using positive value of NRNK as follows: 0 < NRNK <= N-1 :: at most NRNK largest singular values will be used. If the number of the computed nonzero

© 2000-2025, MetaQuotes Ltd.

singular values is less than NRNK, then only those nonzero values will be used and the actually used dimension is less than NRNK.

*tol*

[in] The tolerance for truncating small singular values. 0 <= TOL < 1

*eigen_values*

[out] Vector of eigenvalues of size K. The leading K (K<N) entries of EIGS contain the computed eigenvalues (Ritz values).

*Q*

[out] Orthogonal matrix/factor of the QR factorization of the initial data snapshots matrix F.

*R*

[out] N-by-N upper triangular factor from the QR factorization of the data snapshot matrix F.

*left_vectors*

[out] Matrix of left singular vectors. The leading K columns of X contain the leading K left singular vectors of the hold representations of the leading N-1 snapshots in the orthonormal basis computed in the QR factorization of F. To lift them to the space of the left singular vectors U(:,1:K) of the input data, pre-multiply with the Q factor from the initial QR factorization.

*Z*

[out] M-by-K matrix

If JOBZ =='V' then Z contains the Ritz vectors. Z(:,i) is an eigenvector of the i-th Ritz value; ||Z(:,i)||_2=1.

If JOBZ == 'F', then the Z(:,i)'s are given implicitly as Z*V, where Z contains orthonormal matrix (the product of Q from the initial QR factorization and the SVD/POD_basis returned by GEDMD in X) and the second factor (the eigenvectors of the Rayleigh quotient) is in the matrix V, as returned by GEDMD. That is, X(:,1:K)*V(:,i) is an eigenvector corresponding to EIGS(i). The columns of V(1:K,1:K) are the computed eigenvectors of the K-by-K Rayleigh quotient.

*residuals*

[out] Residuals for the K computed Ritz pairs.

*B*

[out] N-by-K matrix (K<N).

If JOBF =='R', B(1:N,1:K) contains A*U(:,1:K), and can be used for computing the refined vectors.

If JOBF == 'E', B(1:N,1;K) contains A*U(:,1:K)*W(1:K,1:K), which are the vectors from the Exact DMD, up to scaling by the inverse eigenvalues.

In both cases, the content of B can be lifted to the original dimension of the input data by pre-multiplying with the Q factor from the initial QR factorization. Here A denotes a compression of the underlying operator.

If JOBF =='N', then B is not referenced.

*V*

[out] K-by-K matrix (K<N). Contains the the K eigenvectors of the Rayleigh quotient. The Ritz vectors (returned in Z) are the product of Q from the initial QR factorization.

© 2000-2025, MetaQuotes Ltd.

*S*

[out] K-by-K matrix (K<N). The array S(1:K,1:K) is used for the matrix Rayleigh quotient. This content is overwritten during the eigenvalue decomposition by GEEV.

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The number of matrices rows (M) must not be less than the number of columns (N).

### ENUM_DMD_SCALE

An enumeration that determines whether the initial data snapshots are scaled by a diagonal matrix.

| ID | Description |
|---|---|
| DMDSCALE_S | 'S': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that X*D has unit nonzero columns |
| DMDSCALE_C | 'C': The snapshots are scaled as with the 'S' option.If it is found that an i-th column of X is zero vector and the corresponding i-th column of Y is non-zero, then the i-th column of Y is set to zero |
| DMDSCALE_Y | 'Y': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that Y*D has unit nonzero columns |
| DMDSCALE_N | 'N': No data scaling |

### ENUM_DMDQ_EIGV

An enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

| ID | Description |
|---|---|
| DMDQEIGV_V | 'V': The eigenvectors (Koopman modes) will be computed |
| DMDQEIGV_F | 'F': The eigenvectors (Koopman modes) will be returned in factored form as the product Z*V |
| DMDQEIGV_Q | 'Q': The eigenvectors (Koopman modes) will be returned in factored form as the product Q*Z |
| DMDQEIGV_N | 'N': The eigenvectors are not computed |

### ENUM_DMD_RESIDUALS

© 2000-2025, MetaQuotes Ltd.

An enumeration which determines whether to compute the residuals.

| ID | Description |
| --- | --- |
| DMDRESIDUALS_R | 'R': The residuals for the computed eigenpairs will be computed |
| DMDRESIDUALS_N | 'N': The residuals are not computed |

### ENUM_DMDQ_Q

An enumeration which specifies whether to explicitly compute and return the unitary matrix from the QR factorization.

| ID | Description |
| --- | --- |
| DMDQQ_Q | 'Q': The matrix Q of the QR factorization of the data snapshot matrix is computed |
| DMDQQ_N | 'N': The matrix Q is not explicitly computed |

### ENUM_DMDQ_R

An enumeration whichspecifies whether to return the upper triangular factor from the QR factorization.

| ID | Description |
| --- | --- |
| DMDQR_R | 'R': The matrix R of the QR factorization of the data snapshot matrix is computed |
| DMDQR_N | 'N': The matrix R is not explicitly computed |

### ENUM_DMD_REFINE

An enumeration which specifies whether to store information needed for post-processing.

| ID | Description |
| --- | --- |
| DMDREFINE_R | 'R': The matrix needed for the refinement of the Ritz vectors is computed and stored in the array B |
| DMDREFINE_E | 'E': The unscaled eigenvectors of the Exact DMD are computed and returned in the array B |
| DMDREFINE_N | 'N': No eigenvector refinement data is computed |

© 2000-2025, MetaQuotes Ltd.

**ENUM_SVD_ALG**

An enumeration selecting the SVD algorythm.

| ID | Description |
|---|---|
| SVDALG_1 | 1: GESVD (the QR SVD algorithm) |
| SVDALG_2 | 2: GESDD (the Divide and Conquer algorithm) |
| SVDALG_3 | 3: GESVDQ (the preconditioned QR SVD; this and 4 are the most accurate options) |
| SVDALG_4 | 4: GEJSV (the preconditioned Jacobi SVD; this and 3 are the most accurate options) |

© 2000-2025, MetaQuotes Ltd.

# Singular Spectrum Analysis

The section contains functions for decomposing a matrix into three components: orthogonal matrices and a diagonal matrix of singular values. SVD is applied to solve various linear algebra problems such as data dimensionality reduction, image compression, solving systems of equations, as well as data analysis and optimization. The main functions allow calculating singular values and vectors, reconstruct matrices, as well as approximate them with reduced rank accuracy.

| Function | Action |
| --- | --- |
| SingularSpectrumAnalysisSpectrum | A method function for calculating the relative contributions of spectral components based on their eigenvalues. |
| SingularSpectrumAnalysisForecast | A method function for calculating reconstructed and predicted data using spectral components of the input time series. |
| SingularSpectrumAnalysisReconstructComponents | A method function for calculating reconstructed components of the input time series and their contributions. |
| SingularSpectrumAnalysisReconstructSeries | A method function for calculating the reconstructed time series using the first component_count components. |

© 2000-2025, MetaQuotes Ltd.

# SingularSpectrumAnalysisSpectrum

A method function for calculating the relative contributions of spectral components based on their eigenvalues.

### Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisSpectrum(
   ulong   window_length,      // window size for constructing the trajectory matrix
   vector& spectrum            // vector of component contributions to the input serie
   );
```

### Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisSpectrum(
   ulong   window_length,      // window size for constructing the trajectory matrix
   vectorc& spectrum           // vector of component contributions to the input serie
   );
```

### Parameters

*window_length*

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

*spectrum*

[out] Vector of component contributions to the input series - eigenvalues of the covariance matrix of the input time series.

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

© 2000-2025, MetaQuotes Ltd.

# SingularSpectrumAnalysisForecast

A method function for calculating reconstructed and predicted data using spectral components of the input time series.

### Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisForecast(
   ulong    window_length,         // window size for constructing the trajectory mati
   ulong    component_count,       // number of components used for forecasting
   ulong    forecast_horizon,      // number of points to forecast
   vector&  forecast               // vector consisting of reconstructed and predicted
   );
```

### Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisForecast(
   ulong    window_length,         // window size for constructing the trajectory mati
   ulong    component_count,       // number of components used for forecasting
   ulong    forecast_horizon,      // number of points to forecast
   vectorc& forecast               // vector consisting of reconstructed and predicted
   );
```

### Parameters

*window_length*

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

*component_count*

[in] Number of components used for forecasting.

*forecast_horizon*

[in] Number of points to forecast.

*forecast*

[out] Combining points reconstructed by *component_count* plus *forecast_horizon* points predicted using the first *component_count* components. Thus, the *forecast* vector has the size of (T+forecast_horizon), where T is the input series length.

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

© 2000-2025, MetaQuotes Ltd.

# SingularSpectrumAnalysisReconstructComponents

A method function for calculating reconstructed components of the input time series and their contributions.

### Calculations for vector<double> and matrix<double> types

```
bool  vector::SingularSpectrumAnalysisReconstructComponents(
   ulong    window_length,      // window size for constructing the trajectory matrix
   matrix&  components,         // matrix of reconstructed components
   vector&  contributions       // vector of component contributions to the input seri
   );
```

### Calculations for vector<complex> and matrix<complex> types

```
bool  vector::SingularSpectrumAnalysisReconstructComponents(
   ulong    window_length,      // window size for constructing the trajectory matrix
   matrixc& components,         // matrix of reconstructed components
   vectorc& contributions       // vector of component contributions to the input seri
   );
```

### Parameters

*window_length*

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

*components*

[out] A matrix of reconstructed components, where each column describes a separate component.

*contributions*

[out] Vector of component contributions to the input series (eigenvalues of the covariance matrix of the input time series).

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

© 2000-2025, MetaQuotes Ltd.

# SingularSpectrumAnalysisReconstructSeries

A method function for calculating the reconstructed time series using the first *component_count* components.

### Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisReconstructSeries(
   ulong    window_length,       // window size for constructing the trajectory matri
   ulong    component_count,     // number of components used for reconstruction
   vector&  reconstructed        // reconstructed time series
   );
```

### Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisReconstructSeries(
   ulong    window_length,       // window size for constructing the trajectory matri
   ulong    component_count,     // number of components used for reconstruction
   vectorc& reconstructed        // reconstructed time series
   );
```

### Parameters

*window_length*

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

*component_count*

[out] Number of components used for time series reconstruction.

*reconstructed*

[out] Vector containing the reconstructed output series.

### Return Value

The function returns 'true' on success or 'false' if an error occurs.

### Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

© 2000-2025, MetaQuotes Ltd.

# Reductions

Functions designed to transform matrices into special forms using orthogonal transformations. These methods are widely used in numerical computations, such as spectral analysis, eigenvalue problems, and singular value decomposition (SVD) of matrices.

These functions enable efficient matrix transformations essential for solving linear algebra problems, including computing Singular Value Decomposition (SVD) and finding eigenvalues of symmetric matrices. The use of LAPACK functions ensures high performance and computational accuracy.

| Function | Action |
| --- | --- |
| ReduceToBidiagonal | Reduces a general real or complex m-by-n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: Q**T * A * P = B. If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal. Lapack function GEBRD. |
| ReflectBidiagonalToQP | Generates orthogonal matrices Q and P**T (or P**H for complex types) determined by ReduceToBidiagonal method when reducing a real or complex matrix A to bidiagonal form: A = Q * B * P**T. Q and P**T are defined as products of elementary reflectors H(i) or G(i) respectively. Lapack functions ORGBR, UNGBR. |
| ReduceSymmetricToTridiagonal | Reduces a real symmetric or complex Hermitian matrix A to trdiagonal form B by an orthogonal similarity transformation: Q**T * A * Q = B. Lapack functions SYTRD, HETRD. |
| ReflectTridiagonalToQ | Generates orthogonal matrix Q which is defined as the product of n-1 elementary reflectors of order n, as returned by ReduceSymmetricToTridiagonal |

© 2000-2025, MetaQuotes Ltd.

# ReduceToBidiagonal

Reduces a general real or complex m-by-n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: Q**T * A * P = B. If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal. Lapack function GEBRD.

**Computing for type matrix<double>**

```
bool  matrix::ReduceToBidiagonal(
   matrix&          B,             // bidiagonal matrix
   matrix&          reflect_qp,    // q and p-reflectors
   vector&          tau_q,         // scalar factors of the elementary reflectors Q
   vector&          tau_p          // scalar factors of the elementary reflectors P
   );
```

**Computing for type matrix<float>**

```
bool  matrix::ReduceToBidiagonal(
   matrixf&         B,             // bidiagonal matrix
   matrixf&         reflect_qp,    // q and p-reflectors
   vectorf&         tau_q,         // scalar factors of the elementary reflectors Q
   vectorf&         tau_p          // scalar factors of the elementary reflectors P
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::ReduceToBidiagonal(
   matrixc&         B,             // bidiagonal matrix
   matrixc&         reflect_qp,    // q and p-reflectors
   vectorc&         tau_q,         // scalar factors of the elementary reflectors Q
   vectorc&         tau_p          // scalar factors of the elementary reflectors P
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::ReduceToBidiagonal(
   matrixcf&        B,             // bidiagonal matrix
   matrixcf&        reflect_qp,    // q and p-reflectors
   vectorcf&        tau_q,         // scalar factors of the elementary reflectors Q
   vectorcf&        tau_p          // scalar factors of the elementary reflectors P
   );
```

**Parameters**

*B*

[out] Upper or lower bidiagonal matrix.

*reflect_qp*

[out] Transformed matrix A. If m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the vector tau_q, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the vector tau_p, represent the orthogonal matrix P as a product of

© 2000-2025, MetaQuotes Ltd.

elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the vector tau_q, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the vector tau_p, represent the orthogonal matrix P as a product of elementary reflectors.

*tau_q*

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

*tau_p*

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix P.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

The matrices Q and P are represented as products of elementary reflectors:

If m >= n,

Q = H(1) H(2) . . . H(n) and P = G(1) G(2) . . . G(n-1)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v**T and G(i) = I - taup * u * u**T

where tauq and taup are scalars, and v and u are vectors;

v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit in A(i+1:m,i);

u(1:i) = 0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);

tauq is stored in *tau_q*(i) and taup in *tau_p*(i).

If m < n,

Q = H(1) H(2) . . . H(m-1) and P = G(1) G(2) . . . G(m)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v**T and G(i) = I - taup * u * u**T

where tauq and taup are scalars, and v and u are vectors;

v(1:i) = 0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);

u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit in A(i,i+1:n);

tauq is stored in *tau_q*(i) and taup in *tau_p*(i).

The contents of A on exit (output matrix *reflect_qp*) are illustrated by the following examples:

© 2000-2025, MetaQuotes Ltd.

```
m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):
   (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
   (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
   (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
   (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
   (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
   (  v1  v2  v3  v4  v5 )
```

where d and e denote diagonal and off-diagonal elements of B, vi denotes an element of the vector defining H(i), and ui an element of the vector defining G(i).

Matrices Q and P can be produced with ReflectBidiagonalToQP method.

© 2000-2025, MetaQuotes Ltd.

# ReflectBidiagonalToQP

Generates orthogonal matrices Q and P**T (or P**H for complex types) determined by ReduceToBidiagonal method when reducing a real or complex matrix A to bidiagonal form: A = Q * B * P**T. Q and P**T are defined as products of elementary reflectors H(i) or G(i) respectively. Lapack functions ORGBR, UNGBR.

As input is used transformed matrix reflect_qp with the same sizes m-by-n as in original matrix A.

**Computing for type matrix<double>**

```
bool  matrix::ReflectBidiagonalToQP(
   vector&          tau_q,         // scalar factors of the elementary reflectors Q
   vector&          tau_p,         // scalar factors of the elementary reflectors P
   matrix&          Q,             // matrix Q
   matrix&          PT             // transposed matrix P
   );
```

**Computing for type matrix<float>**

```
bool  matrix::ReflectBidiagonalToQP(
   vectorf&         tau_q,         // scalar factors of the elementary reflectors Q
   vectorf&         tau_p,         // scalar factors of the elementary reflectors P
   matrixf&         Q,             // matrix Q
   matrixf&         PT             // transposed matrix P
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::ReflectBidiagonalToQP(
   vectorc&         tau_q,         // scalar factors of the elementary reflectors Q
   vectorc&         tau_p,         // scalar factors of the elementary reflectors P
   matrixc&         Q,             // matrix Q
   matrixc&         PH             // hermitian conjugated matrix P
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::ReflectBidiagonalToQP(
   vectorcf&        tau_q,         // scalar factors of the elementary reflectors Q
   vectorcf&        tau_p,         // scalar factors of the elementary reflectors P
   matrixcf&        Q,             // matrix Q
   matrixcf&        PH             // hermitian conjugated matrix P
   );
```

**Parameters**

*tau_q*

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

*tau_p*

© 2000-2025, MetaQuotes Ltd.

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix P.

*Q*

[out] Orthogonal matrix Q.

*PT (PH)*

[out] Transposed (or hermitian conjugated) matrix P.

### Return Value

Return true if successful, otherwise false in case of an error.

### Note

If m >= n, matrix Q is of m-by-n sizes, matrix PT is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix PT is of m-by-n sizes.

© 2000-2025, MetaQuotes Ltd.

# ReduceSymmetricToTridiagonal

Reduces a real symmetric or complex Hermitian matrix A to trdiagonal form B by an orthogonal similarity transformation: Q**T * A * Q = B. Lapack functions SYTRD, HETRD.

### Computing for type matrix<double>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrix&         B,             // tridiagonal matrix
   matrix&         reflect_q,     // q-reflectors
   vector&         tau_q          // scalar factors of the elementary reflectors Q
   );
```

### Computing for type matrix<float>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixf&        B,             // tridiagonal matrix
   matrixf&        reflect_q,     // q-reflectors
   vectorf&        tau_q          // scalar factors of the elementary reflectors Q
   );
```

### Computing for type matrix<complex>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixc&        B,             // tridiagonal matrix
   matrixc&        reflect_q,     // q-reflectors
   vectorc&        tau_q          // scalar factors of the elementary reflectors Q
   );
```

### Computing for type matrix<complexf>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixcf&       B,             // tridiagonal matrix
   matrixcf&       reflect_q,     // q and p-reflectors
   vectorcf&       tau_q          // scalar factors of the elementary reflectors Q
   );
```

### Parameters

*B*

[out] Symmetric (or Hermitian) tridiagonal matrix.

*reflect_q*

[out] Upper or lower triangular matrix, it depends on the input matrix A. In upper case the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix B, and the elements above the first superdiagonal, with the array *tau_q*, represent the orthogonal matrix Q as a product of elementary reflectors; in lower case the diagonal and first subdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix B, and the elements below the first subdiagonal, with the array *tau_q*, represent the orthogonal matrix Q as a product of elementary reflectors.

*tau_q*

© 2000-2025, MetaQuotes Ltd.

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

**Return Value**

Return true if successful, otherwise false in case of an error.

**Note**

In upper case the matrix Q is represented as a product of elementary reflectors

Q = H(n-1) . . . H(2) H(1)

Each H(i) has the form

H(i) = I - tau * v * v**T

where tau is a scalar, and v is a vector with v(i+1:n) = 0 and v(i) = 1; v(1:i-1) is stored on exit in A(1:i-1,i+1), and tau in *tau_q*(i).

In lower case the matrix Q is represented as a product of elementary reflectors

Q = H(1) H(2) . . . H(n-1)

Each H(i) has the form

H(i) = I - tau * v * v**T

where tau is a scalar, and v is a vector with v(1:i) = 0 and v(i+1) = 1; v(i+2:n) is stored on exit in A(i+2:n,i), and tau in *tau_q*(i). */

The contents of A on exit (output matrix *reflect_q*) are illustrated by the following examples with n = 5:

```
      upper case:                          lower case:

      (  d   e   v2  v3  v4 )              (  d                   )
      (      d   e   v3  v4 )              (  e   d               )
      (          d   e   v4 )              (  v1  e   d           )
      (              d   e  )              (  v1  v2  e   d       )
      (                  d  )              (  v1  v2  v3  e   d   )
```

where d and e denote diagonal and off-diagonal elements of B and vi denotes an element of the vector defining H(i).

Matrix Q can be produced with ReflectTridiagonalToQ method.

© 2000-2025, MetaQuotes Ltd.

# ReflectTridiagonalToQ

Generates orthogonal matrix Q which is defined as the product of n-1 elementary reflectors of order n, as returned by ReduceSymmetricToTridiagonal:

in upper case Q = H(n-1) . . . H(2) H(1),

in lower case Q = H(1) H(2) . . . H(n-1).

Lapack functions ORGTR, UNGTR.

As input is used transformed matrix reflect_q with the same sizes n-by-n as in original matrix A.

**Computing for type matrix<double>**

```
bool  matrix::ReflectTridiagonalToQ(
   vector&          tau_q,         // scalar factors of the elementary reflectors Q
   matrix&          Q              // matrix Q
   );
```

**Computing for type matrix<float>**

```
bool  matrix::ReflectTridiagonalToQ(
   vectorf&         tau_q,         // scalar factors of the elementary reflectors Q
   matrixf&         Q              // matrix Q
   );
```

**Computing for type matrix<complex>**

```
bool  matrix::ReflectTridiagonalToQ(
   vectorc&         tau_q,         // scalar factors of the elementary reflectors Q
   matrixc&         Q              // matrix Q
   );
```

**Computing for type matrix<complexf>**

```
bool  matrix::ReflectTridiagonalToQ(
   vectorcf&        tau_q,         // scalar factors of the elementary reflectors Q
   matrixcf&        Q              // matrix Q
   );
```

**Parameters**

*tau_q*

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

*Q*

[out] Orthogonal matrix Q.

**Return Value**

Return true if successful, otherwise false in case of an error.

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# 函数转换

函数组提供从一组数据到另一种数据的转换。

NormalizeDouble()函数可以特殊标识，它提供必要的价值描述精确度。在交易操作中，如果交易服务器超过了字节要求的精确度，不能使用非标准价格。

| 函数 | 功能 |
| --- | --- |
| CharToString | 转换符号代码到字符串 |
| DoubleToString | 以规定的精确度转换数字值到文本行 |
| EnumToString | 将任何类型的枚举值转换为字符串 |
| NormalizeDouble | 浮点数转换到规定的精确度 |
| StringToDouble | 转换包括数字代表符的字符串到双精度数字 |
| StringToInteger | 转换包括数字代表符的字符串到整型数字 |
| StringToTime | 转换包括 "yyyy.mm.dd [hh:mi]"格式的时间或者日期字符串到日期时间型 |
| TimeToString | 转换01.01.1970值到 "yyyy.mm.dd hh:mi"字符串格式 |
| IntegerToString | 整型转换成当前长度字符串 |
| ShortToString | 转换符号代码 (双字节编码)到单字符串 |
| ShortArrayToString | 复制数组部分到字符串 |
| StringToShortArray | 符号-wise复制字符串到选定的无符号短整型部分数组 |
| CharArrayToString | 转换符号代码 (ansi)到单符号数组 |
| StringToCharArray | 符号-wise 复制双字节编码到ANSI转换的字符串，到选定的无符字符型数组 |
| CharArrayToStruct | 将uchar类型数组复制到POD结构 |
| StructToCharArray | 将POD结构复制到uchar类型数组 |
| ColorToARGB | 转换color类型到uint类型，接收ARGB颜色显示 |
| ColorToString | 转换颜色值到"R,G,B"形式的字符串 |
| StringToColor | 转换"R,G,B"字符串或者颜色名的字符串到颜色类型值 |
| StringFormat | 按照现格式转换数字到字符串 |

另见

代码页使用

© 2000-2025, MetaQuotes Ltd.

# CharToString

将字符型转换成字符串型结果。

```
string  CharToString(
   uchar  char_code      // 交易品种的数值代码
   );
```

常量

*char_code*

[in] 统一码交易品种代码

返回值

统一码交易品种的字符串。

示例：

```
string ExtStrArray[224];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在从交易品种代码32 (空格)到255 (UCHAR _ MAX)的循环中,
//--- 根据当前代码页,用转换为字符串的交易品种代码填充数组
   for(int i=32; i<=UCHAR_MAX; i++)
      ExtStrArray[i-32]=CharToString((uchar)i);

//--- 在日志中记录标题和交易品种表格
   Print("Table of symbols:");
   ArrayPrint(ExtStrArray,_Digits," | ");
   /*
   result:
   Table of symbols:
   [  0] " " | "!" | """ | "#" | "$" | "%" | "&" | "'" | "(" | ")" | "*" | "+" | ","
   [ 25] "9" | ":" | ";" | "<" | "=" | ">" | "?" | "@" | "A" | "B" | "C" | "D" | "E"
   [ 50] "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "[" | "\" | "]" | "^"
   [ 75] "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w"
   [100] "„" | "…" | "†" | "‡" | "€" | "‰" | "Љ" | "‹" | "Њ" | "Ќ" | "Ћ" | "Џ" | "Ђ"
   [125] "ќ" | "ћ" | "џ" | " " | "ў" | "Ў" | "J" | "¤" | "Ґ" | "¦" | "§" | "Ё" | "©"
   [150] "¶" | "·" | "ё" | "№" | "є" | "»" | "ј" | "S" | "s" | "ї" | "A" | "Б" | "B"
   [175] "П" | "Р" | "С" | "Т" | "У" | "Ф" | "Х" | "Ц" | "Ч" | "Ш" | "Щ" | "Ъ" | "Ы"
   [200] "и" | "й" | "к" | "л" | "м" | "н" | "о" | "п" | "р" | "с" | "т" | "у" | "ф"
   */
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

StringToCharArray、ShortToString、StringGetCharacter

© 2000-2025, MetaQuotes Ltd.

# CharArrayToString

将双字符型部分数组复制或转换成返回型字符串

```
string  CharArrayToString(
   uchar  array[],               // 数组
   int    start=0,               // 数组启动位置
   int    count=-1               // 交易品种数
   uint    codepage=CP_ACP       // 代码页
   );
```

常量

*array[]*

[in] 双字符型数组

*start=0*

[in] 复制开始位置，默认值是0

*count=-1*

[in] 复制的数组元素数量，定义结果字符串长度，默认值是-1，代表从数组末尾开始复制，或者从0终端开始

*codepage=CP_ACP*

[in] 代码页值，为最多使用的 code pages 提供适当常量。

返回值

字符串

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建带有交易品种代码的数组
   uchar array[]= { 84, 104, 105, 115,  32, 105, 115,  32,  97,  32,
                    116, 101, 115, 116,  32, 111, 102,  32, 116, 104,
                    101,  32,  67, 104,  97, 114,  65, 114, 114,  97,
                    121,  84, 111,  83, 116, 114, 105, 110, 103,  40,
                     41,  32, 102, 117, 110,  99, 116, 105, 111, 110
                   };
//--- 在日志中记录转换为字符串的代码数组
   Print(CharArrayToString(array));

//--- 创建带有交易品种代码和程序端为零的数组
   uchar array_z[]= { 84, 104, 105, 115,  32, 105, 115,  32,  97,  32,
                      116, 101, 115, 116,   0,  32, 111, 102,  32, 116,
```

© 2000-2025, MetaQuotes Ltd.

```
                     104, 101,  32,  67, 104,  97, 114,  65, 114, 114,
                      97, 121,  84, 111,  83, 116, 114, 105, 110, 103,
                      40,  41,  32, 102, 117, 110,  99, 116, 105, 111, 110
                    };
//--- 在日志中记录程序端为零的代码第二数组
   PrintFormat("%s ...", CharArrayToString(array_z));

   /*
   result:
   This is a test of the CharArrayToString() function
   This is a test ...
   */
  }
```

另见

StringToCharArray、ShortArrayToString、代码页使用

© 2000-2025, MetaQuotes Ltd.

# CharArrayToStruct

将uchar类型数组复制到POD结构。

```
bool  CharArrayToStruct(
   void&          struct_object,     //结构
   const uchar&   char_array[],      //数组
   uint           start_pos=0        // 数组中的起始位置
   );
```

参数

*struct_object*

[in] 参考任何类型的POD结构（只包含简单的数据类型）。

*char_array[]*

[in] uchar类型数组。

*start_pos=0*

[in] 数组中的位置，从这里开始复制数据。

返回值

如果成功返回true，否则返回false。

示例：

```
#define DATA_TOTAL 4

MqlRates ExtDataRates[DATA_TOTAL];
uchar    ExtCharArray[];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 将最后四个柱形图的数据复制到uchar数组
   ResetLastError();
   for(int i=0; i<DATA_TOTAL; i++)
     {
      if(!CopyRatesToCharArray(i, ExtCharArray))
         return;
     }

//--- 按照uchar数组中的数据量，将循环中的所有可用数据复制到MqlRates结构数组中
   for(int i=0; i<DATA_TOTAL; i++)
     {
      ResetLastError();
      if(!CharArrayToStruct(ExtDataRates[i], ExtCharArray, sizeof(MqlRates)*i))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("CharArrayToStruct() failed. Error code: ", GetLastError());
       return;
      }
    //--- 在成功地将数据从uchar数组复制到MqlRates结构后，
    //--- 在日志中记录复制到结构中的数据
    MqlRatesPrint(ExtDataRates[i]);
   }

/*
result:
MqlRates data:
 Time = 2024.02.21 09:00;
 Open = 1.08116;
 High = 1.08141;
 Low = 1.08062;
 Close = 1.08124;
 Tick volume = 5344;
 Spread = 1;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 08:00;
 Open = 1.08149;
 High = 1.08153;
 Low = 1.08073;
 Close = 1.08114;
 Tick volume = 3607;
 Spread = 2;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 07:00;
 Open = 1.08143;
 High = 1.08165;
 Low = 1.08122;
 Close = 1.08150;
 Tick volume = 2083;
 Spread = 0;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 06:00;
 Open = 1.08178;
 High = 1.08190;
 Low = 1.08132;
 Close = 1.08142;
 Tick volume = 1733;
 Spread = 0;
```

© 2000-2025, MetaQuotes Ltd.

```
    Real volume = 0
   */
  }
//+------------------------------------------------------------------+
//| 将柱形图数据复制到uchar数组                                      |
//| 通过MqlRates结构的指定索引                                       |
//+------------------------------------------------------------------+
bool CopyRatesToCharArray(const int index, uchar &data_array[])
  {
   MqlRates rates[1];
   uint     start=sizeof(MqlRates);

   ResetLastError();
   if(CopyRates(Symbol(), PERIOD_CURRENT, index, 1, rates)!=1)
     {
      Print("CopyRates() failed. Error code: ", GetLastError());
      return(false);
     }
   if(!StructToCharArray(rates[0], data_array, start*index))
     {
      Print("StructToCharArray() failed. Error code: ", GetLastError());
      return(false);
     }

   return(true);
  }
//+------------------------------------------------------------------+
//| 在日志中记录MqlRates结构的字段                                   |
//+------------------------------------------------------------------+
void MqlRatesPrint(MqlRates &rates)
  {
//--- 创建输出字符串
   string text=StringFormat(" Time = %s;\n"
                            " Open = %.*f;\n"
                            " High = %.*f;\n"
                            " Low = %.*f;\n"
                            " Close = %.*f;\n"
                            " Tick volume = %I64u;\n"
                            " Spread = %d;\n"
                            " Real volume = %I64u",
                            TimeToString(rates.time),
                            _Digits, rates.open,
                            _Digits, rates.high,
                            _Digits, rates.low,
                            _Digits, rates.close,
                            rates.tick_volume,
                            rates.spread,
                            rates.real_volume);
//--- 在日志中记录标题和输出字符串
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("MqlRates data:\n", text, "\n");
  }
```

另见

StringToCharArray、ShortArrayToString、StructToCharArray、使用代码页、FileReadStruct、Unions (union)、MathSwap

© 2000-2025, MetaQuotes Ltd.

# StructToCharArray

将POD结构复制到uchar类型数组。

```
bool  StructToCharArray(
   const void&  struct_object,      //结构
   uchar&       char_array[],       // 数组
   uint         start_pos=0         // 数组中的起始位置
   );
```

参数

*struct_object*

[in] 参考任何类型的POD结构（只包含简单的数据类型）。

*char_array[]*

[in] uchar类型数组。

*start_pos=0*

[in] 数组中的位置，从这里开始添加复制的数据。

返回值

如果成功返回true，否则返回false。

注意

当复制时，如果没有足够的空间，动态数组将自动展开(ArrayResize)。如果数组不能展开到所需的值，该函数将返回一个错误。

示例：

```
uchar    ExtCharArray[];
MqlRates ExtRates[1];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 将当前柱形图的数据复制到MqlRates结构
   if(CopyRates(Symbol(), PERIOD_CURRENT, 0, 1, ExtRates)!=1)
     {
      Print("CopyRates() failed. Error code: ", GetLastError());
      return;
     }

//--- 在日志中记录MqlRates结构中接收的数据
   Print("Data in the structure immediately after receiving it:");
   MqlRatesPrint(ExtRates[0]);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 将该结构复制到uchar类型数组中
   ResetLastError();
   if(!StructToCharArray(ExtRates[0], ExtCharArray))
     {
      Print("StructToCharArray() failed. Error code: ", GetLastError());
      return;
     }

//--- 清空结构
   ZeroMemory(ExtRates[0]);
//--- 清空后在结构中记录数据
   Print("\nData in the structure after ZeroMemory():");
   MqlRatesPrint(ExtRates[0]);

//--- 现在将数据从uchar数组复制到MqlRates结构
   if(!CharArrayToStruct(ExtRates[0], ExtCharArray))
     {
      Print("CharArrayToStruct() failed. Error code: ", GetLastError());
      return;
     }
//--- 将数据从uchar数组复制后记录到MqlRates结构中
   Print("\nData in the MqlRates structure after copying it from a uchar array:");
   MqlRatesPrint(ExtRates[0]);

   /*
   result:

   Data in the structure immediately after receiving it:
   MqlRates data:
    Time = 2024.02.21 07:00;
    Open = 1.08143;
    High = 1.08158;
    Low = 1.08122;
    Close = 1.08137;
    Tick volume = 1341;
    Spread = 4;
    Real volume = 0

   Data in the structure after ZeroMemory():
   MqlRates data:
    Time = 0;
    Open = 0.00000;
    High = 0.00000;
    Low = 0.00000;
    Close = 0.00000;
    Tick volume = 0;
    Spread = 0;
    Real volume = 0
```

© 2000-2025, MetaQuotes Ltd.

```
   Data in the MqlRates structure after copying it from a uchar array:
   MqlRates data:
    Time = 2024.02.21 07:00;
    Open = 1.08143;
    High = 1.08158;
    Low = 1.08122;
    Close = 1.08137;
    Tick volume = 1341;
    Spread = 4;
    Real volume = 0
   */
  }
//+------------------------------------------------------------------+
//| 在日志中记录MqlRates结构的字段                                   |
//+------------------------------------------------------------------+
void MqlRatesPrint(MqlRates &rates)
  {
//--- 创建输出字符串
   string text=StringFormat(" Time = %s;\n"
                            " Open = %.*f;\n"
                            " High = %.*f;\n"
                            " Low = %.*f;\n"
                            " Close = %.*f;\n"
                            " Tick volume = %I64u;\n"
                            " Spread = %d;\n"
                            " Real volume = %I64u",
                            TimeToString(rates.time),
                            _Digits, rates.open,
                            _Digits, rates.high,
                            _Digits, rates.low,
                            _Digits, rates.close,
                            rates.tick_volume,
                            rates.spread,
                            rates.real_volume);
//--- 在日志中记录标题和输出字符串
   Print("MqlRates data:\n", text);
  }
```

另见

StringToCharArray、ShortArrayToString、CharArrayToStruct、使用代码页、FileWriteStruct、Unions (union)、MathSwap

© 2000-2025, MetaQuotes Ltd.

# ColorToARGB

该函数将color 类型转换为uint 类型以获得ARGB的颜色显示。ARGB颜色格式被用于生成图形资源, 文本展示，以及被用在CCanvas标准程序库类。

```
uint  ColorToARGB(
   color  clr,          // 以color格式转换颜色
   uchar  alpha=255     // 阿尔法通道管理彩色透明度
   );
```

参数

*clr*

[in] color类型变量的颜色值。

*alpha*

[in] 阿尔法通道的值用来接收ARGB格式的颜色。该值可以从0 (前景像素颜色不改变根本显示的颜色)设置到255 (基本像素的颜色被前景像素颜色替代)。彩色透明度计算为 (1-alpha/255)*100%。换句话说，阿尔法通道的值越小颜色越透明。

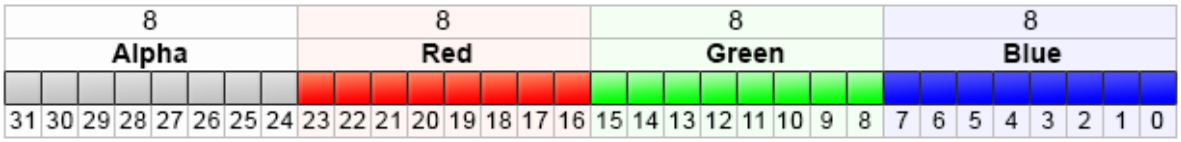
返回值

以ARGB格式显示颜色，在这里Alfa, Red, Green, Blue (阿尔法通道，红，绿，蓝)的值按照四种uint类型字节系列设置。

注意

RGB 是基本的普通的使用格式，用于电脑图形屏幕的像素颜色描述。基本颜色名称通常用于红，绿和蓝色组件。每个组件通过在0-255 (16进制格式0x00 到 0XFF)范围内指定色彩饱和度字节进行描述。由于白色涵盖了所有颜色，它被描述为0xFFFFFF，那就是，0xFF最大值呈现三个组件的每一个。

然而，如果它以带有透明度的颜色覆盖，那么一些任务需要指定色彩透明度以便描述图形。阿尔法通道就是为此引入的。它作为RGB格式的额外组件实施。ARGB格式结构显示如下。




ARGB值通常使用16禁止的格式来表达，每组数字分别代表阿尔法，红色，绿色和蓝色通道的值。例如，80FFFF00颜色代表透明度50.2% 的黄色。最初，0x80 设置 50.2% 阿尔法值，因为0xFF值的50.2%。然后，第一组FF定义红色组件的最高值，下一组FF类似前一个但是绿色；最后组00代表蓝色组件的最低值 (没有蓝色)。绿色和红色合成黄色。如果不使用阿尔法通道，输入可能减到6 RRGGBB 数字，这就是将阿尔法通道值存储在uint整型最高位的原因。

根据内容，16进制数字可以通过'0x' 或 '#' prefix编写，例如，80FFFF00, 0x80FFFF00 或 #80FFFF00。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置透明度
   uchar alpha=0x55;  // 0x55 means 55/255=21.6 % of transparency
   //--- 派生转换clrBlue 到 ARGB
   PrintFormat("0x%.8X - clrBlue",clrBlue);
   PrintFormat("0x%.8X - clrBlue ARGB with alpha=0x55 (transparency 21.6%%)",ColorToAR
   //--- 派生转换clrGreen 到 ARGB
   PrintFormat("0x%.8X - clrGreen",clrGreen);
   PrintFormat("0x%.8X - clrGreen ARGB with alpha=0x55 (transparency 21.6%%)",ColorToA
   //--- 派生转换clrRed color 到 ARGB
   PrintFormat("0x%.8X - clrRed",clrRed);
   PrintFormat("0x%.8X - clrRed ARGB with alpha=0x55 (transparency 21.6%%)",ColorToARG
  }
```

另见

资源, ResourceCreate(), TextOut(), 颜色类型, char, short, int 和长整型

© 2000-2025, MetaQuotes Ltd.

# ColorToString

转换颜色值到字符串RGB形式

```
string  ColorToString(
   color  color_value,     // 颜色值
   bool   color_name       // 是否显示颜色名称
   );
```

常量

*color_value*

[in] 颜色类型变量颜色值

*color_name*

[in] 如果颜色名称与预定义的 颜色常量之一相同，有必要返回颜色名称标签。

返回值

字符串颜色描述成RGB，RGB小数位常量从0到255转变成字符串。如果设置color_name=true参量，就会把颜色值转变成颜色名称

示例：

```
   string clr=ColorToString(C'0,255,0'); // 绿色
   Print(clr);

   clr=ColorToString(C'0,255,0',true);   // 获得颜色常量
   Print(clr);
```

另见

StringToColor, ColorToARGB

© 2000-2025, MetaQuotes Ltd.

# DoubleToString

把数字值转换成文本串

```
string  DoubleToString(
   double  value,      // 数字
   int     digits=8    // 小数点后的数字数
   );
```

参量

*value*

[in] 浮点值

*digits*

[in]精确模式。如果***digits***值在0到16之间，字符串描述数值是在小数点后特定的数字。如果***digits***值在-1到-16之间，字符串描述数值是小数点后的标准数值。在其他任何情况下在小数点后保留8位。

返回值

字符串包括交易品种代表规定精确度数量。

示例：

```
   Print("DoubleToString(120.0 + M_PI) : ",DoubleToString(120.0+M_PI));
   Print("DoubleToString(120.0 + M_PI,16) : ",DoubleToString(120.0+M_PI,16));
   Print("DoubleToString(120.0 + M_PI,-16) : ",DoubleToString(120.0+M_PI,-16));
   Print("DoubleToString(120.0 + M_PI,-1) : ",DoubleToString(120.0+M_PI,-1));
   Print("DoubleToString(120.0 + M_PI,-20) : ",DoubleToString(120.0+M_PI,-20));
```

另见

NormalizeDouble, StringToDouble

© 2000-2025, MetaQuotes Ltd.

# EnumToString

转换任何类型的枚举值到文本格式。

```
string  EnumToString(
   any_enum  value      // 任何类型的枚举值
   );
```

参数

*value*

[in] 任何类型的枚举值。

返回值

枚举文本显示字符串。若要获得错误信息，请调用GetLastError() 函数。

注意

该函数可以在LastError变量设置下列错误值：

- ERR_INTERNAL_ERROR -执行环境错误
- ERR_NOT_ENOUGH_MEMORY -完成操作内存不足
- ERR_INVALID_PARAMETER -不允许枚举值名称

例如：

```
enum interval  // 命名常量的枚举
  {
   month=1,     // 一个月间隔
   two_months,  // 两个月
   quarter,     // 三个月 - 一季度
   halfyear=6,  // 半年
   year=12,     // 一年 - 12 个月
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置等于一个月的时间间隔
   interval period=month;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- 设置等于一个季度 (三个月)的时间间隔
   period=quarter;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- 设置等于一年 (12个月)的时间间隔
   period=year;
```

© 2000-2025, MetaQuotes Ltd.

```
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- 检查订单类型如何显示
   ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;
   Print(EnumToString(type)+"="+IntegerToString(type));

//--- 检查错误值如何显示
   type=WRONG_VALUE;
   Print(EnumToString(type)+"="+IntegerToString(type));

// 结果:
// 月份=1
// 季度=3
// 年=12
// ORDER_TYPE_BUY=0
// ENUM_ORDER_TYPE::-1=-1
  }
```

另见

枚举, 输入变量

© 2000-2025, MetaQuotes Ltd.

# IntegerToString

函数将整数类型值转变成特定长度的字符串值并返回包含的字符串。

```
string  IntegerToString(
   long     number,              // 数字
   int      str_len=0,           // 字符串结果长度
   ushort   fill_symbol=' '      // 填充物
   );
```

参量

*数字*

[in] 转换数量

*str_len=0*

[in] 字符串长度。如果结果字符串长度超过指定长度，字符串不会切断，如果短，从左边添加交易品种。

*fill_symbol=' '*

[in] 填补交易品种，默认分开

返回值

字符串。

示例：

```
#define DATA_TOTAL 1001

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在DATA_TOTAL循环中显示索引倍数为100的行
//--- 该字符串以四个字符格式显示索引值
//--- 如果字符串长度少于4个字符，则字符串中循环索引的值
//--- 前面以零开头
   for(int i=0; i<DATA_TOTAL; i++)
     {
      if(i%100==0)
         Print("Converted index value: ",IntegerToString(i,4,'0'));
     }
   /*
   result:
   Converted index value: 0000
   Converted index value: 0100
   Converted index value: 0200
   Converted index value: 0300
```

© 2000-2025, MetaQuotes Ltd.

```
   Converted index value: 0400
   Converted index value: 0500
   Converted index value: 0600
   Converted index value: 0700
   Converted index value: 0800
   Converted index value: 0900
   Converted index value: 1000
   */
  }
```

另见

StringToInteger

© 2000-2025, MetaQuotes Ltd.

# ShortToString

把交易品种代码（统一码）转变成字符串交易品种中的一中，返回结果字符串。

```
string  ShortToString(
   ushort  symbol_code      // 交易品种
   );
```

参量

*symbol_code*

[in] 交易品种代码，可以使用包含交易品种或2字节十六进制代码的文本字符串代替交易品种代码，但在同一码平台上与交易品种代码相一致。

返回值

字符串。

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在循环中记录18个字符，从Unicode编码为U+23E9的字符开始
   for(int i=0; i<18; i++)
     {
      ushort code=0x23E9+(ushort)i;
      PrintFormat("Unicode number U+%hX: %s",code,ShortToString(code));
     }
   /*
   result:
   Unicode number U+23E9: ⏩
   Unicode number U+23EA: ⏪
   Unicode number U+23EB: ⏫
   Unicode number U+23EC: ⏬
   Unicode number U+23ED: ⏭
   Unicode number U+23EE: ⏮
   Unicode number U+23EF: ⏯
   Unicode number U+23F0: ⏰
   Unicode number U+23F1: ⏱
   Unicode number U+23F2: ⏲
   Unicode number U+23F3: ⏳
   Unicode number U+23F4:
   Unicode number U+23F5:
   Unicode number U+23F6:
   Unicode number U+23F7:
   Unicode number U+23F8: ⏸
```

© 2000-2025, MetaQuotes Ltd.

```
   Unicode number U+23F9: ⏹
   Unicode number U+23FA: ⏺
   */
  }
```

另见

StringToCharArray、CharToString、StringGetCharacter

© 2000-2025, MetaQuotes Ltd.

# ShortArrayToString

把数组的复制部分转到返回字符串。

```
string  ShortArrayToString(
   ushort  array[],      // 数组
   int     start=0,      // 数组中的启动位置
   int     count=-1      // 交易品种数
   );
```

参量

*array[]*

[in] 无符号短整型数组（模拟wchar_t型）

*start=0*

[in] 位置，从复制开始起，默认是0

*count=-1*

[in] 复制的数组元素数字，定义结果字符串的长度，默认值是-1，表示复制到数组末尾，否则客户端仍旧是0

返回值

字符串。

示例：

```
#define ROW_SIZE 16

ushort ExtShortArray[]={0x2190,0x2191,0x2192,0x2193,0x2194,0x2195,0x2196,0x2197,0x2198
                        0x21A0,0x21A1,0x21A2,0x21A3,0x21A4,0x21A5,0x21A6,0x21A7,0x21A8
                        0x21B0,0x21B1,0x21B2,0x21B3,0x21B4,0x21B5,0x21B6,0x21B7,0x21B8
                        0x21C0,0x21C1,0x21C2,0x21C3,0x21C4,0x21C5,0x21C6,0x21C7,0x21C8
                        0x21D0,0x21D1,0x21D2,0x21D3,0x21D4,0x21D5,0x21D6,0x21D7,0x21D8
                        0x21E0,0x21E1,0x21E2,0x21E3,0x21E4,0x21E5,0x21E6,0x21E7,0x21E8
                        0x21F0,0x21F1,0x21F2,0x21F3,0x21F4,0x21F5,0x21F6,0x21F7,0x21F8

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 计算7x16交易品种表中的字符串数量
   int total=(int)ExtShortArray.Size()/ROW_SIZE;
//--- 在表格行的循环中，显示7组箭头，每行16个Unicode字符，逐行显示到日志中
   for(int i=0; i<total; i++)
     {
      int row=i*ROW_SIZE;
      PrintFormat("Arrow set %u: %s",i+1,ShortArrayToString(ExtShortArray,row,ROW_SIZE
```

© 2000-2025, MetaQuotes Ltd.

```
   }
  /*
  result:
  Arrow set 1: ←↑→↓↔↕□□□□□□□□□□
  Arrow set 2: □□□□□□□□↕□□□□□□□
  Arrow set 3: □□□□□□□□□□□□□□□□
  Arrow set 4: □□□□□□□□□□□□□□□□
  Arrow set 5: □□⇒□⇔□□□□□□□□□□
  Arrow set 6: □□□□□□□□□□□□□□□□
  Arrow set 7: □□□□□□□□□□□□□□□□
  */
 }
```

另见

StringToShortArray、CharArrayToString、代码页使用

© 2000-2025, MetaQuotes Ltd.

# TimeToString

转变值包括从01.01.1970起已消耗的秒数，以字符串格式"yyyy.mm.dd hh:mi"

```
string  TimeToString(
   datetime  value,                          // 数字
   int       mode=TIME_DATE|TIME_MINUTES     // 输出形式
   );
```

参量

*值*

[in] 时间从00:00 1970/01/01开始

*mode=TIME_DATE|TIME_MINUTES*

[in]额外数据输入模式，可以是一个也可以包括以
"yyyy.mm.dd"为形式的标签TIME_DATE，
以"hh:mi"为结果的 TIME_MINUTES，
以 "hh:mi:ss"为结果的TIME_SECONDS 。

返回值

字符串。

示例：

```
datetime ExtBarTimeOpen;

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 将timer设置为秒
   EventSetTimer(1);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator deinitialization function                       |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   Comment("");
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
```

© 2000-2025, MetaQuotes Ltd.

```
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---获取当前柱形图的开仓时间
   ExtBarTimeOpen=time[rates_total-1];
//--- 为下一次调用返回前次已计算值
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Timer函数                                                        |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- 设置之前柱形图开仓时间
   static datetime bar_open_time=ExtBarTimeOpen;
//--- 记录柱形图开仓后经过的秒数
   static int seconds=int(TimeCurrent()-ExtBarTimeOpen);
//--- 如果之前的开仓时间不等于当前时间，则这是一个新柱形图
//--- 将新开仓时间写为之前的开仓时间，并将秒数设置为零
   if(bar_open_time!=ExtBarTimeOpen)
     {
      bar_open_time=ExtBarTimeOpen;
      seconds=0;
     }
//--- 增加并调整自柱形图开仓以来经过的秒数
   seconds++;
   if(seconds>PeriodSeconds(PERIOD_CURRENT))
      seconds=0;
//--- 柱形图开仓时间为yyyy.mm.dd hh:mi
   string bar_time_open=TimeToString(ExtBarTimeOpen);
//--- 当前时间为yyyy.mm.dd hh:mi:ss
   string time_current=TimeToString(TimeCurrent(),TIME_DATE|TIME_MINUTES|TIME_SECONDS)
//--- 距离新柱形图开仓还剩多少秒
   int    sec_left=PeriodSeconds(PERIOD_CURRENT)-seconds;
//--- 当前柱形图关闭前剩余的时间，格式为 hh:mm:ss
   string time_left=TimeToString(sec_left,TIME_MINUTES|TIME_SECONDS);
//--- 创建输出字符串
   string txt=StringFormat("Opening time of the current bar: %s\n"+
                           "Time Current: %s\n"+
                           "Seconds have passed since the bar opened: %d\n"+
                           "Approximately seconds left before bar closes: %d\n"+
                           "Time remaining until bar closes: %s",bar_time_open,time_cu
//--- 显示柱形图开仓时间和当前时间，
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 显示当前柱形图开仓到关闭所经过的秒数，
//--- 在注释中显示当前柱线关闭前剩余的时间
   Comment(txt);
   /*
   result on M1:
   Opening time of the current bar: 2024.02.22 18:06
   Time Current: 2024.02.22 18:06:24
   Seconds have passed since the bar opened: 25
   Approximately seconds left before bar closes: 35
   Time remaining until bar closes: 00:00:35

   result on M5:
   Opening time of the current bar: 2024.02.22 18:05
   Time Current: 2024.02.22 18:07:28
   Seconds have passed since the bar opened: 149
   Approximately seconds left before bar closes: 151
   Time remaining until bar closes: 00:02:31

   result on H1:
   Opening time of the current bar: 2024.02.22 18:00
   Time Current: 2024.02.22 18:08:13
   Seconds have passed since the bar opened: 494
   Approximately seconds left before bar closes: 3106
   Time remaining until bar closes: 00:51:46

   result on D1:
   Opening time of the current bar: 2024.02.22 00:00
   Time Current: 2024.02.22 18:11:01
   Seconds have passed since the bar opened: 65462
   Approximately seconds left before bar closes: 20938
   Time remaining until bar closes: 05:48:58
   */
  }
```

另见

StringToTime、TimeToStruct

© 2000-2025, MetaQuotes Ltd.

# NormalizeDouble

把浮点类型转变成指定精确度

```
double  NormalizeDouble(
   double   value,      // 标准化号码
   int      digits      // 小数点后的数字数
   );
```

参量

*value*

[in] 浮点类型值

*digits*

[in] 精确度模式，浮点数字 (0-8).

返回值

预先设置的双精度类型值

注释

待办订单止损数值，目标数值和开仓值的计算值一定根据精确度规范吗，该值可以通过 Digits()获得。

请注意，当使用Print()函数输出到日志时，标准数字可能包括比你想象的更多的小数位。例如：

```
   double a=76.671;              // 三个小数位的标准数字
   Print("Print(76.671)=",a);    // 输出为
   Print("DoubleToString(a,8)=",DoubleToString(a,8)); // 以预设精度输出
```

在程序端您将拥有以下信息：

```
DoubleToString(a,8)=76.67100000

Print(76.671)=76.67100000000001
```

示例：

```
   double pi=M_PI;
   Print("pi = ",DoubleToString(pi,16));

   double pi_3=NormalizeDouble(M_PI,3);
   Print("NormalizeDouble(pi,3) = ",DoubleToString(pi_3,16))
   ;
   double pi_8=NormalizeDouble(M_PI,8);
   Print("NormalizeDouble(pi,8) = ",DoubleToString(pi_8,16));

   double pi_0=NormalizeDouble(M_PI,0);
   Print("NormalizeDouble(pi,0) = ",DoubleToString(pi_0,16));
```

© 2000-2025, MetaQuotes Ltd.

```
/*
   Result:
   pi= 3.1415926535897931
   NormalizeDouble(pi,3)= 3.1419999999999999
   NormalizeDouble(pi,8)= 3.1415926499999998
   NormalizeDouble(pi,0)= 3.0000000000000000
*/
```

另见 so

DoubleToString, 真实型 双精度，浮点）, 类型减少

© 2000-2025, MetaQuotes Ltd.

# StringToCharArray

交易品种复制从一个Unicode型字符串转变成ANSI，双字符型数组的选择位置，返回复制元素的数量。

```
int  StringToCharArray(
   string  text_string,          // 源字符串
   uchar&  array[],              // 数组
   int     start=0,              // 数组中的启动位置
   int     count=-1              // 交易品种数
   uint    codepage=CP_ACP       // 代码页
   );
```

参量

*text_string*

[in] 复制字符串型

*array[]*

[out] 无符字符型数组

*start=0*

[in] 从起始位置开始复制，默认值是0

*count=-1*

[in] 数组元素复制数量，定义字符串结果的长度，默认值是-1，会一直复制到数组的结尾，或者是终端0。终端0也会被接收数组复制，在此情形下，如果字符串大小需要，动态数组的大小是能增加的，如果动态数组增大到跟路线一样长，数组的体积不会减小。

*codepage=CP_ACP*

[in] 代码页的值，最多使用 代码页 提供适当常量

返回值

复制元素的数量

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 要转换的字符串
   string text = "This is a test of the StringToCharArray() function";

//--- 根据设置的代码页将输入字符串转换为uchar数组
   uchar char_array[];
   int   copied=StringToCharArray(text, char_array);
   PrintFormat("String length: %u\nNumber of characters copied (with terminal 0): %d\r
```

© 2000-2025, MetaQuotes Ltd.

```
                StringLen(text), copied, text);
//--- 在日志中记录结果数组
   ArrayPrint(char_array, 0, " | ");
   /*
   result:
   String length: 50
   Number of characters copied (with terminal 0): 51
   Array of characters for the string 'This is a test of the StringToCharArray() funct
   [ 0]  84 | 104 | 105 | 115 |  32 | 105 | 115 |  32 |  97 |  32 | 116 | 101 | 115 |
   [17]  32 | 116 | 104 | 101 |  32 |  83 | 116 | 114 | 105 | 110 | 103 |  84 | 111 |
   [34]  65 | 114 | 114 |  97 | 121 |  40 |  41 |  32 | 102 | 117 | 110 |  99 | 116 |
   */
  }
```

另见

CharArrayToString、StringToShortArray、代码页使用

© 2000-2025, MetaQuotes Ltd.

# StringToColor

使用颜色名称将RGB字符串型转变成颜色类型值

```
color  StringToColor(
   string  color_string      // 字符串颜色表示
   );
```

参量

*color_string*

[in] 代表颜色的RGB类型字符串类型或者其中一种预定义 网页-颜色 名称

返回值

颜色值

示例：

```
   color str_color=StringToColor("0,127,0");
   Print(str_color);
   Print((string)str_color);
//--- 变换颜色
   str_color=StringToColor("0,128,0");
   Print(str_color);
   Print((string)str_color);
```

另见

ColorToString, ColorToARGB

© 2000-2025, MetaQuotes Ltd.

# StringToDouble

将字符串转变成数字的双精度型

```
double  StringToDouble(
   string  value      // 字符串
   );
```

参量

*值*

[in] 数字的字符串换行格式

返回值

双精度型值

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 要转换的字符串
   string str = "12345.54321";
//--- 将输入字符串作为实数转换为double型变量
   double converted=StringToDouble(str);
//--- 在日志中显示精确到小数点后8位的结果数字
   PrintFormat("The string '%s' is converted to the real number %.8f", str, converted)
   /*
   result:
   The string '12345.54321' is converted to the real number 12345.54321000
   */
  }
```

另见

NormalizeDouble、真实型（双精度，浮点）、类型减少

© 2000-2025, MetaQuotes Ltd.

# StringToInteger

将字符串型转换成整型结果。

```
long  StringToInteger(
   string  value      // 字符串
   );
```

参量

*值*

[in] 数字的字符串转换格式

返回值

整型值

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 要转换的字符串
   string str = "12345.54321";
//--- 将输入字符串作为数字转换为长整型变量
   long converted=StringToInteger(str);
//--- 在日志中显示获得的数字，小数部分被截断
   PrintFormat("The string '%s' is converted to the integer number %I64d", str, convei
   /*
   result:
   The string '12345.54321' is converted to the integer number 12345
   */
  }
```

另见

IntegerToString、真实型（双精度，浮点）、类型减少

© 2000-2025, MetaQuotes Ltd.

# StringToShortArray

函数的交易品种复制一组字符串到长短型数组的规定地点，返回复制元素的数量。

```
int  StringToShortArray(
   string  text_string,      // 源字符串
   ushort& array[],          // 数组
   int     start=0,          // 数组启动位置
   int     count=-1          // 交易品种数
   );
```

参量

*text_string*

[in] 字符串复制

*array[]*

[out] 无符短型 型数组 (wchar_t模拟类型)

*start=0*

[in] 位置，从复制值开始，默认值是0

*count=-1*

[in] 复制的数组元素的数量，定义字符串结果的长度，默认值是-1，意味着根据数组的结尾来进行复制，直到0是终点。终点0也会复制到接收数组，在此情形下，如果字符串大小需要，动态数组的大小是能增加的，如果动态数组增大到跟路线一样长，数组的体积不会减小。

返回值

复制元素的数量

示例：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 要转换的chars字符串
   string text = "Chars: ♡♫☎✈☩☏ ❁✉☆ ⌓♕ ♠®✧ ❄ ♣    ";
   ushort short_array[];
//--- 在输入字符串中找到“:”字符的位置并从下一个字符开始提取子字符串
   int    pos=StringFind(text, ":");
   string str=(pos<0 ? text : StringSubstr(text, pos+1));
//--- 从结果字符串的左侧和右侧删除空格、回车符和制表符
   StringTrimLeft(str);
   StringTrimRight(str);
//--- 将结果字符串复制到ushort数组，并将结果数组记录到日志中
   int copied=StringToShortArray(str, short_array);
```

© 2000-2025, MetaQuotes Ltd.

```
   PrintFormat("String of characters length: %u\n"
               "Number of characters copied (with terminal 0): %d\n"
               "Array of chars for the string '%s':",
               StringLen(str), copied, str);
   ArrayPrint(short_array, 0, " | ");
   /*
   result:
   String of characters length: 16
   Number of characters copied (with terminal 0): 17
   Array of chars for the string '♡♫☎☄✣☏❁✉☆☁♕♠®✧❄♣':
   10084 |  9835 |  9742 |  9992 | 10019 |  9743 |  9728 |  9993 |  9734 |  9729 |  98
   */
  }
```

另见

ShortArrayToString、StringToCharArray、代码页使用

© 2000-2025, MetaQuotes Ltd.

# StringToTime

将"yyyy.mm.dd [hh:mi]"格式中包含时间和/或日期的字符串转换格式为datetime类型号。

```
datetime  StringToTime(
   const string  time_string      // 日期字符串
   );
```

参数

*time_string*

[in] 其中一种指定格式的字符串：

- "yyyy.mm.dd [hh:mi]"
- "yyyy.mm.dd [hh:mi:ss]"
- "yyyymmdd [hh:mi:ss]"
- "yyyymmdd [hhmiss]"
- "yyyy/mm/dd [hh:mi:ss]"
- "yyyy-mm-dd [hh:mi:ss]"

返回值

datetime类型值，包含自01.01.1970以来过去的秒数。

注意

为了避免在调用StringToTime()之前对*time_string*进行额外处理，可以将日期和时间之间的任何空间和制表符序列视为单个空间。

示例：

```
//--- 输入参数
input group    "The date can be entered in any of the formats:"
input group    "yyyy.mm.dd [hh:mi], yyyy.mm.dd [hh:mi:ss]"
input group    "yyyymmdd [hh:mi:ss], yyyymmdd [hhmiss]"
input group    "yyyy/mm/dd [hh:mi:ss], yyyy-mm-dd [hh:mi:ss]"
input string   InpDateStr;    // Please enter the date here as a string

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 将在输入中以字符串形式输入的时间转换为datetime值
   datetime time=StringToTime(InpDateStr);
//--- 在日志中显示输入的字符串和获得的时间
   PrintFormat("Date entered as a string in the form '%s' is converted to datetime in
               InpDateStr, TimeToString(time, TIME_DATE|TIME_MINUTES|TIME_SECONDS));
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在收到的日期时间上创建一条垂直线，并将图表移至此位置
   if(CreateVLine(time))
      ChartNavigateToTime(time);
   /*
   result:
   Date entered as a string in the form '' is converted to datetime in the form '1970.
   Date entered as a string in the form '2024' is converted to datetime in the form '2
   Date entered as a string in the form '202400' is converted to datetime in the form
   Date entered as a string in the form '20240000' is converted to datetime in the for
   Date entered as a string in the form '2024022410' is converted to datetime in the
   Date entered as a string in the form '20240224 10' is converted to datetime in the
   Date entered as a string in the form '20240224 01' is converted to datetime in the
   Date entered as a string in the form '20240224 0030' is converted to datetime in th
   Date entered as a string in the form '20240224 0100' is converted to datetime in th
   */
  }
//+------------------------------------------------------------------+
//| Create a vertical line object                                    |
//+------------------------------------------------------------------+
bool CreateVLine(const datetime line_time)
  {
   ResetLastError();

   string name=MQLInfoString(MQL_PROGRAM_NAME)+"_VLINE";
   if(!ObjectCreate(0, name, OBJ_VLINE, 0, line_time, 0))
     {
      Print("ObjectCreate() failed. Error code: ", GetLastError());
      return(false);
     }
   ObjectSetInteger(0, name, OBJPROP_STYLE, STYLE_DOT);
   ObjectSetInteger(0, name, OBJPROP_SELECTABLE, true);

   return(true);
  }
//+------------------------------------------------------------------+
//| Shift the chart to the specified bar opening time                |
//+------------------------------------------------------------------+
bool ChartNavigateToTime(const datetime time)
  {
   ChartSetInteger(0, CHART_AUTOSCROLL, false);
   ResetLastError();

   int bar=iBarShift(_Symbol, PERIOD_CURRENT, time);
   if(bar<0)
     {
      PrintFormat("%s: iBarShift() failed. Error code: %d", __FUNCTION__, GetLastErro
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   long first=0;
   if(!ChartGetInteger(0, CHART_FIRST_VISIBLE_BAR, 0, first))
     {
      PrintFormat("%s: ChartGetInteger() failed. Error code: %d", __FUNCTION__, GetLas
      return(false);
     }

   return(ChartNavigate(0, CHART_CURRENT_POS, (int)first-bar));
  }
```

另见

TimeToString、TimeToStruct

© 2000-2025, MetaQuotes Ltd.

# StringFormat

该函数格式包括参量和字符串的返回

```
string  StringFormat(
   string  format,      // 带有格式描述的字符串
   ...     ...          // 参量
   );
```

参数

*format*

[in] 字符串包含格式化办法，格式化规则与 PrintFormat 函数相同。

*...*

[in] 参量，被逗号分开。

返回值

字符串。

示例：

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
//--- 字符串变量
   string output_string;
   string temp_string;
   string format_string;
//--- 准备规范标题
   temp_string=StringFormat("Contract specification for %s:\n",_Symbol);
   StringAdd(output_string,temp_string);
//--- 整型值输出
   int digits=(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   temp_string=StringFormat("   SYMBOL_DIGITS = %d (number of digits after the decimal
                            digits);
   StringAdd(output_string,temp_string);
//--- 小数点后可变数位的双精度值输出
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   format_string=StringFormat("   SYMBOL_POINT = %%.%df (point value)\n",
                              digits);
   temp_string=StringFormat(format_string,point_value);
   StringAdd(output_string,temp_string);
//--- 整型值输出
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   temp_string=StringFormat("   SYMBOL_SPREAD = %d (current spread in points)\n",
                            spread);
   StringAdd(output_string,temp_string);
//--- 整型值输出
   int min_stop=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
   temp_string=StringFormat("   SYMBOL_TRADE_STOPS_LEVEL = %d (minimal indention in po
                            min_stop);
   StringAdd(output_string,temp_string);
//--- 无小数部分的双精度值输出
   double contract_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_CONTRACT_SIZE);
   temp_string=StringFormat("   SYMBOL_TRADE_CONTRACT_SIZE = %.f (contract size)\n",
                            contract_size);
   StringAdd(output_string,temp_string);
//--- 默认精度的双精度值输出
   double tick_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_TICK_SIZE);
   temp_string=StringFormat("   SYMBOL_TRADE_TICK_SIZE = %f (minimal price change)\n",
                            tick_size);
   StringAdd(output_string,temp_string);
//--- 决定库存费计算模式
   int swap_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_SWAP_MODE);
   string str_swap_mode;
   switch(swap_mode)
     {
      case SYMBOL_SWAP_MODE_DISABLED: str_swap_mode="SYMBOL_SWAP_MODE_DISABLED (no swa
      case SYMBOL_SWAP_MODE_POINTS: str_swap_mode="SYMBOL_SWAP_MODE_POINTS (in points)
      case SYMBOL_SWAP_MODE_CURRENCY_SYMBOL: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
      case SYMBOL_SWAP_MODE_CURRENCY_MARGIN: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
      case SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY
      case SYMBOL_SWAP_MODE_INTEREST_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST
      case SYMBOL_SWAP_MODE_INTEREST_OPEN: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST_OP
      case SYMBOL_SWAP_MODE_REOPEN_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_CUR
      case SYMBOL_SWAP_MODE_REOPEN_BID: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_BID (by
     }
//--- 字符串值输出
   temp_string=StringFormat("   SYMBOL_SWAP_MODE = %s\n",
                            str_swap_mode);
   StringAdd(output_string,temp_string);
//--- 默认精度的双精度值输出
   double swap_long=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_LONG);
```

© 2000-2025, MetaQuotes Ltd.

```
   temp_string=StringFormat("   SYMBOL_SWAP_LONG = %f (long swap value)\n",
                            swap_long);
   StringAdd(output_string,temp_string);
//--- 默认精度的双精度值输出
   double swap_short=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_SHORT);
   temp_string=StringFormat("   SYMBOL_SWAP_SHORT = %f (short swap value)\n",
                            swap_short);
   StringAdd(output_string,temp_string);
//--- 决定交易模式
   int trade_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_MODE);
   string str_trade_mode;
   switch(trade_mode)
     {
      case SYMBOL_TRADE_MODE_DISABLED: str_trade_mode="SYMBOL_TRADE_MODE_DISABLED (tra
      case SYMBOL_TRADE_MODE_LONGONLY: str_trade_mode="SYMBOL_TRADE_MODE_LONGONLY (onl
      case SYMBOL_TRADE_MODE_SHORTONLY: str_trade_mode="SYMBOL_TRADE_MODE_SHORTONLY (o
      case SYMBOL_TRADE_MODE_CLOSEONLY: str_trade_mode="SYMBOL_TRADE_MODE_CLOSEONLY (o
      case SYMBOL_TRADE_MODE_FULL: str_trade_mode="SYMBOL_TRADE_MODE_FULL (no trade re
     }
//--- 字符串值输出
   temp_string=StringFormat("   SYMBOL_TRADE_MODE = %s\n",
                            str_trade_mode);
   StringAdd(output_string,temp_string);
//--- 压缩格式的双精度值输出
   double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   temp_string=StringFormat("   SYMBOL_VOLUME_MIN = %g (minimal volume for a deal)\n",
   StringAdd(output_string,temp_string);
//--- 压缩格式的双精度值输出
   double volume_step=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_STEP);
   temp_string=StringFormat("   SYMBOL_VOLUME_STEP = %g (minimal volume change step)\n
   StringAdd(output_string,temp_string);
//--- 压缩格式的双精度值输出
   double volume_max=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MAX);
   temp_string=StringFormat("   SYMBOL_VOLUME_MAX = %g (maximal volume for a deal)\n",
   StringAdd(output_string,temp_string);
//--- 确定合约价格计算模式
   int calc_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_CALC_MODE);
   string str_calc_mode;
   switch(calc_mode)
     {
      case SYMBOL_CALC_MODE_FOREX:str_calc_mode="SYMBOL_CALC_MODE_FOREX (Forex)";breal
      case SYMBOL_CALC_MODE_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_FUTURES (futures)"
      case SYMBOL_CALC_MODE_CFD:str_calc_mode="SYMBOL_CALC_MODE_CFD (CFD)";break;
      case SYMBOL_CALC_MODE_CFDINDEX:str_calc_mode="SYMBOL_CALC_MODE_CFDINDEX (CFD for
      case SYMBOL_CALC_MODE_CFDLEVERAGE:str_calc_mode="SYMBOL_CALC_MODE_CFDLEVERAGE (C
      case SYMBOL_CALC_MODE_EXCH_STOCKS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_STOCKS (t
      case SYMBOL_CALC_MODE_EXCH_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FUTURES
      case SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FU
     }
//--- 字符串值输出
   temp_string=StringFormat("   SYMBOL_TRADE_CALC_MODE = %s\n",
                            str_calc_mode);
   StringAdd(output_string,temp_string);
//--- 小数点后两位的双精度值输出
   double margin_initial=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_INITIAL);
   temp_string=StringFormat("   SYMBOL_MARGIN_INITIAL = %.2f (initial margin)\n",
                            margin_initial);
   StringAdd(output_string,temp_string);
//--- 小数点后两位的双精度值输出
   double margin_maintenance=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_MAINTENANCE);
   temp_string=StringFormat("   SYMBOL_MARGIN_MAINTENANCE = %.2f (maintenance margin)\
```

© 2000-2025, MetaQuotes Ltd.

```
                              margin_maintenance);
   StringAdd(output_string,temp_string);
//--- 整型值输出
   int freeze_level=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_FREEZE_LEVEL);
   temp_string=StringFormat("   SYMBOL_TRADE_FREEZE_LEVEL = %d (order freeze level in
                              freeze_level);
   StringAdd(output_string,temp_string);
   Print(output_string);
   Comment(output_string);
/* 执行结果
    EURUSD的合约规范:
     SYMBOL_DIGITS = 5 (小数点后的数位)
     SYMBOL_POINT = 0.00001 (点值)
     SYMBOL_SPREAD = 10 (当前点差的点)
     SYMBOL_TRADE_STOPS_LEVEL = 18 (止损订单的最小缩进点)
     SYMBOL_TRADE_CONTRACT_SIZE = 100000 (合约大小)
     SYMBOL_TRADE_TICK_SIZE = 0.000010 (最小价格变动)
     SYMBOL_SWAP_MODE = SYMBOL_SWAP_MODE_POINTS (点)
     SYMBOL_SWAP_LONG = -0.700000 (买入订单的库存费的值)
     SYMBOL_SWAP_SHORT = -1.000000 (卖出订单的库存费的值)
     SYMBOL_TRADE_MODE = SYMBOL_TRADE_MODE_FULL (无交易限制)
     SYMBOL_VOLUME_MIN = 0.01 (一笔成交的最小交易量)
     SYMBOL_VOLUME_STEP = 0.01 (最小交易量变化步骤)
     SYMBOL_VOLUME_MAX = 500 (一笔成交的最大交易量)
     SYMBOL_TRADE_CALC_MODE = SYMBOL_CALC_MODE_FOREX (外汇)
     SYMBOL_MARGIN_INITIAL = 0.00 (初始预付款)
     SYMBOL_MARGIN_MAINTENANCE = 0.00 (维持预付款)
     SYMBOL_TRADE_FREEZE_LEVEL = 0 (订单冻结水平点)
*/
  }
```

另见

PrintFormat, DoubleToString,ColorToString, TimeToString

© 2000-2025, MetaQuotes Ltd.

# 数学函数

数学函数和三角函数

数学函数最初设计用于对标量值执行相关操作。在此基础上，大多数函数都可以应用于矩阵和向量。这些包括MathAbs、MathArccos、MathArcsin、MathArctan、MathCeil、MathCos、MathExp、MathFloor、MathLog、MathLog10、MathMod、MathPow、MathRound、MathSin、MathSqrt、MathTan、MathExpm1、MathLog1p、MathArccosh、MathArcsinh、MathArctanh、MathCosh、MathSinh和MathTanh。这类操作意味着对矩阵或向量进行元素式处理。示例：

```
//---
  matrix a= {{1, 4}, {9, 16}};
  Print("matrix a=\n",a);
  a=MathSqrt(a);
  Print("MatrSqrt(a)=\n",a);
  /*
   matrix a=
    [[1,4]
     [9,16]]
   MatrSqrt(a)=
    [[1,2]
     [3,4]]
  */
```

对于MathMod和MathPow，第二个元素既可以是标量，也可以是相应大小的矩阵/向量。

| 函数 | 功能 |
| --- | --- |
| MathAbs | 返回指定数值的绝对值（绝对值） |
| MathArccos | 返回x弧度的反余弦 |
| MathArcsin | 返回x弧度的反正弦 |
| MathArctan | 返回x弧度的反正切 |
| MathArctan2 | 返回其正切为两个指定数值的商的弧度角 |
| MathClassify | 返回实数型 |
| MathCeil | 从上面返回最靠近的整数数值 |
| MathCos | 返回数字余弦 |
| MathExp | 返回数字指数 |
| MathFloor | 从下面返回最靠近的整数数值 |
| MathLog | 返回自然对数 |
| MathLog10 | 以10为底返回对数 |
| MathMax | 返回两个数值的最大值 |
| MathMin | 返回两个数值的最小值 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| MathMod | 两个数值相除后返回实余数 |
| MathPow | 从基数升到额定输出 |
| MathRand | 0-32767范围返回随机值 |
| MathRound | 四舍五入到最近整数值 |
| MathSin | 返回正弦数 |
| MathSqrt | 返回平方根 |
| MathSrand | 设置开始点生成系列随机数 |
| MathTan | 返回正切数 |
| MathIsValidNumber | 检测实数真实型 |
| MathExpm1 | 返回MathExp(x)-1表达式值 |
| MathLog1p | 返回MathLog(1+x)表达式值 |
| MathArccosh | 返回双曲反余弦 |
| MathArcsinh | 返回双曲反正弦 |
| MathArctanh | 返回双曲反正切 |
| MathCosh | 返回双曲余弦 |
| MathSinh | 返回双曲正弦 |
| MathTanh | 返回双曲正切 |
| MathSwap | 在ushort/uint/ushort类型值中改变字节的顺序 |

© 2000-2025, MetaQuotes Ltd.

# MathAbs

函数返回特定数值的绝对值 绝对值）

```
double  MathAbs(
   double  value      // 数值
   );
```

参量

*值*

[in] 数值

返回值

双精度类型值多于或等于0

注释

取代MathAbs()函数，可以使用 fabs()。

示例：

```
//--- 输入参数
input int      InpValue = -10;   // 在这里输入任何值

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取输入中输入数字的绝对值
   uint abs_value=MathAbs(InpValue);
//--- 在日志中记录值
   PrintFormat("The entered value %d received the value %u after using the MathAbs() f
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathArccos

函数在0到 π弧度返回x。

```
double  MathArccos(
   double  val     // -1<val<1
   );
```

参量

*val*

[in] val值在-1到1之间，计算反余弦

返回值

编号幅度反余弦，如果val小于-1或者大于1，函数返回NaN (不确定值)。

注释

取代MathArccos()函数可以使用 acos()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/1000);
   delta[0]=-1;
//--- 获取从-1到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的反余弦值
   vector Y=MathArccos(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

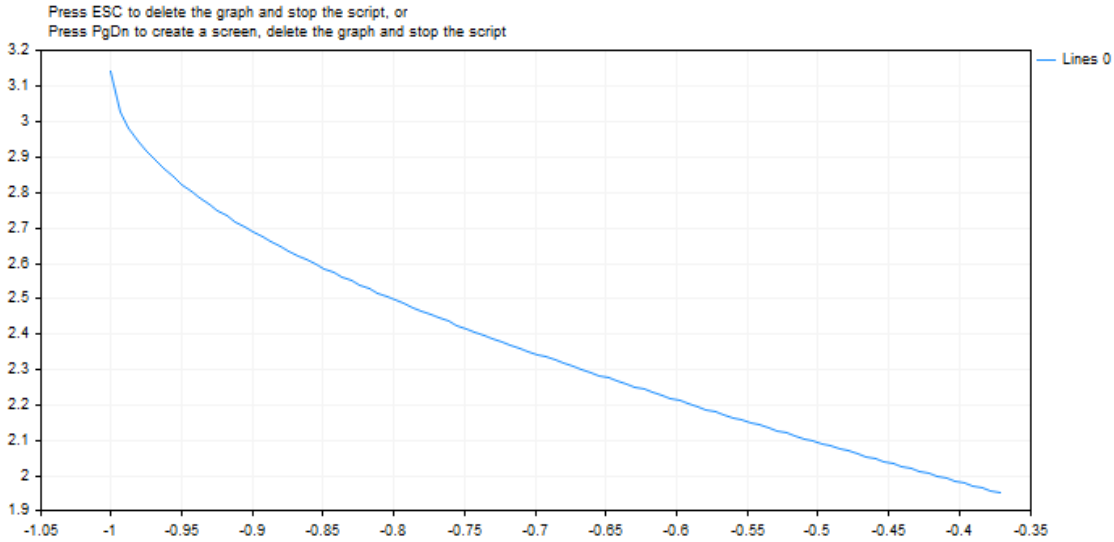
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and saves the image to a file                  |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathArcsin

函数余弦x在-π/2 到 π/2范围之内

```
double  MathArcsin(
   double  val      // -1<值<1
   );
```

参量

*val*

[in] va l值在-1到1之间，计算余弦值

返回值

va l数的余弦在-π/2 到 π/2 之间，如果va l小于-1，或者大于1，函数返回值是NaN (不确定值)

注释

取代MathArcsin()函数，可以使用 asin()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/1000);
   delta[0]=-1;
//--- 获取从-1到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的正弦值
   vector Y=MathArcsin(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

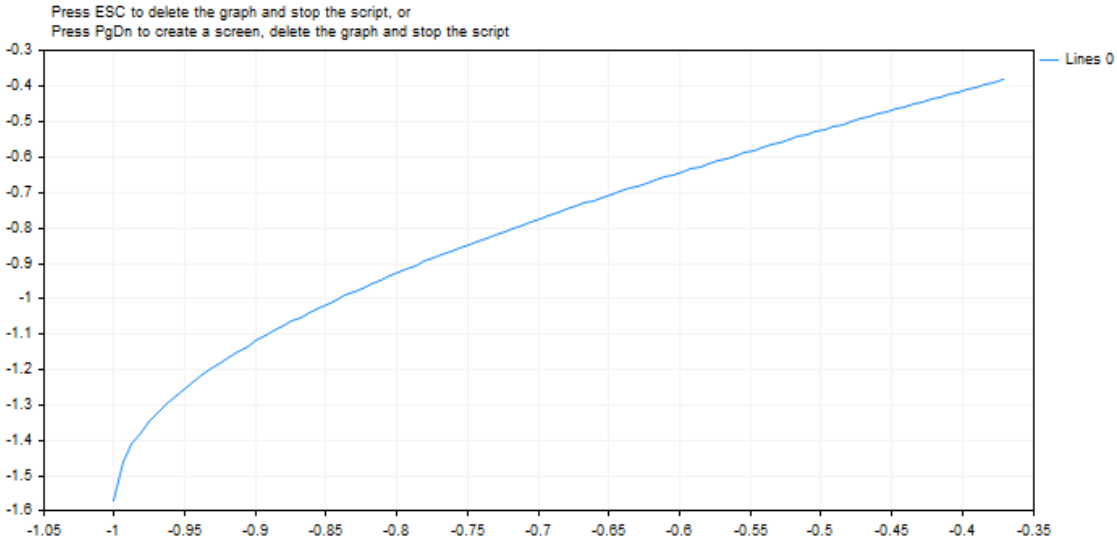
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathArctan

函数返回反正切x，如果x等于0，函数返回0

```
double  MathArctan(
   double  value      // 正切
   );
```

参量

*值*

[in] 数字代表正切

返回值

MathArctan返回值在-π/2 到 π/2之间。

注释

代替MathArctan()函数，可以使用 atan()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的反正切值
   vector Y=MathArctan(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

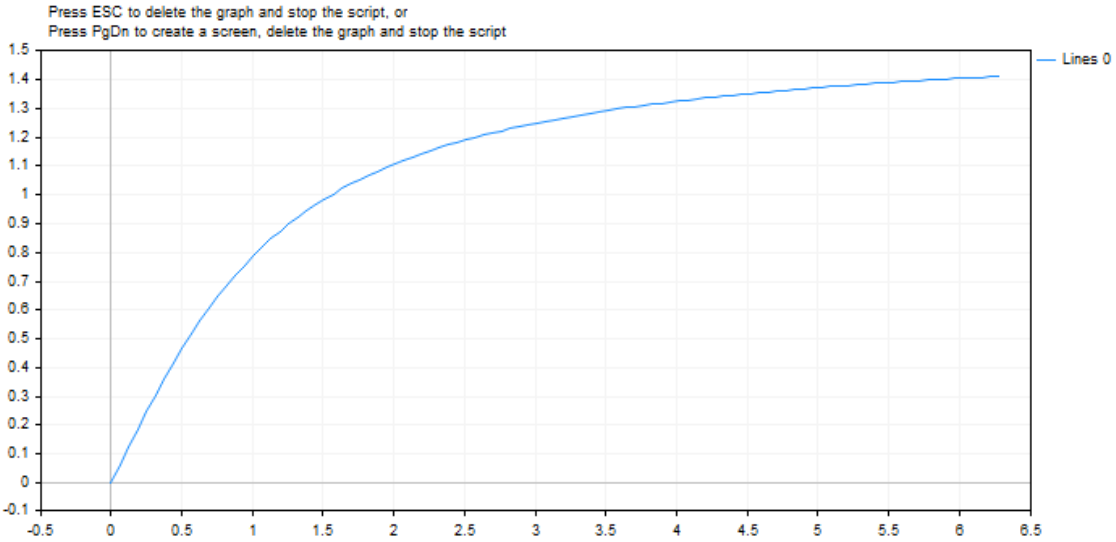
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathArctan2

返回其正切为两个指定数值的商的弧度角。

```
double MathArctan2(
 double y      // 点的y坐标
 double x      // 点的x坐标
  );
```

参数

*y*

[in] Y坐标值。

*x*

[in] X坐标值。

返回值

MathArctan2返回一个角，θ，其范围弧度从 -π到π，因此MathTan(θ)=y/x。

请注意以下情况：

- (x, y)在象限1内，0< θ< π/2
- (x, y)在象限2内，π/2< θ ≤ π
- (x, y)在象限3内，-π < θ < -π/2
- (x, y)在象限4内，-π/2 < θ < 0

对于象限边界的点，返回值如下：

- 如果 y 等于 0 且 x 不是负数，那么 θ= 0。
- 如果 y 等于 0 且x 为负数，那么 θ= π。
- 如果 y 等于正数，而 x 等于 0，那么 θ= π/2。
- 如果 y 等于负数，而 x 等于 0，那么 θ= -π/2。
- 如果 y 等于 0 且 x 也是0，那么 θ= 0.

注意

您可以使用atan2()函数，代替MathArctan2()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

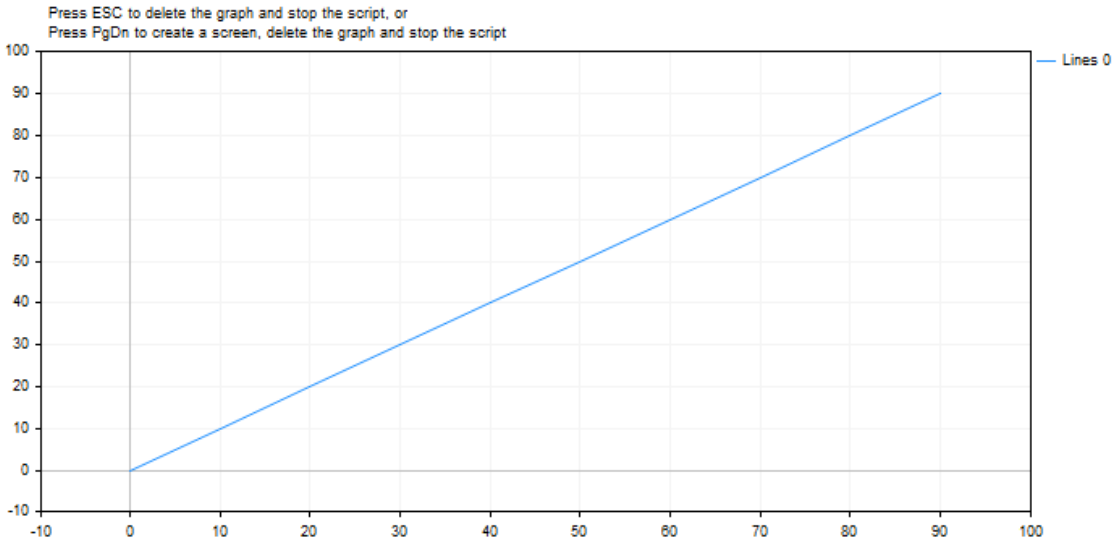
```
  {
   vector delta=vector::Full(10,10);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的反正切值
   vector Y=delta.CumSum();

   Print("vector delta = \n",delta);
   Print("vector X = \n",X);
   Print("vector Y = \n",Y);

//--- 将计算值从向量转移到数组
   double x_array[];;
   double y_array[];;
   X.Swap(x_array);
   Y.Swap(y_array);

   double array[10];
   for(int i=0; i<10; i++)
     {
      array[i]=MathArctan2(y_array[i],x_array[i]);
     }

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
```

© 2000-2025, MetaQuotes Ltd.

```
    if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
       return(true);
//--- 返回'false'
    return(false);
   }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
   {
    ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
    ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
    ExtGraph.IndentUp(30);
    ExtGraph.CurvePlotAll();
    string text1="Press ESC to delete the graph and stop the script, or";
    string text2="Press PgDn to create a screen, delete the graph and stop the script";
    ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
    ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
    ExtGraph.Update();
   }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
   {
    string file_names[];
    ResetLastError();
    int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
    if(selected<1)
      {
       if(selected<0)
          PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
       return false;
      }

    bool res=false;
    if(ChartSetInteger(0,CHART_SHOW,false))
       res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
    ChartSetInteger(0,CHART_SHOW,true);
    return(res);
   }
```

结果：

© 2000-2025, MetaQuotes Ltd.




© 2000-2025, MetaQuotes Ltd.

# MathClassify

确定实数型，并以ENUM_FP_CLASS枚举值的形式返回结果

```
ENUM_FP_CLASS  MathClassify(
   double  value      // 实数
   );
```

参数

*值*

[in] 要被检查的实数

返回值

来自ENUM_FP_CLASS枚举的值

**ENUM_FP_CLASS**

| ID | 描述 |
|---|---|
| FP_SUBNORMAL | 比最小可表示正规数DBL_MIN (2.2250738585072014e-308)更接近于零的次正规数 |
| FP_NORMAL | 正规数的范围在2.2250738585072014e-308与1.7976931348623158e+308之间 |
| FP_ZERO | 正零或负零 |
| FP_INFINITE | 无法用适当的类型（正或负无穷大）表示的数字 |
| FP_NAN | 非数字。 |

例如：

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- test NaN
  double nan=double("nan");
  PrintFormat("Test NaN: %G is %s, MathIsValidNumber(NaN)=%s",
              nan,
              EnumToString(MathClassify(nan)),
              (string)MathIsValidNumber(nan));
//--- test infinity
  double inf=double("inf");
  PrintFormat("Test Inf: %G is %s, MathIsValidNumber(inf)=%s",
              inf,
              EnumToString(MathClassify(inf)),
```

© 2000-2025, MetaQuotes Ltd.

```
               (string)MathIsValidNumber(inf));
//--- test normal value
  double normal=1.2345e6;
  PrintFormat("Test Normal: %G is %s, MathIsValidNumber(normal)=%s",
               normal,
               EnumToString(MathClassify(normal)),
               (string)MathIsValidNumber(normal));
//--- test subnormal value
  double sub_normal=DBL_MIN/2.0;
  PrintFormat("Test Subnormal: %G is %s, MathIsValidNumber(sub_normal)=%s",
               sub_normal,
               EnumToString(MathClassify(sub_normal)),
               (string)MathIsValidNumber(sub_normal));
//--- test zero value
  double zero=0.0/(-1);
  PrintFormat("Test Zero: %G is %s, MathIsValidNumber(zero)=%s",
               zero,
               EnumToString(MathClassify(zero)),
               (string)MathIsValidNumber(zero));
 }
 /*
 Result:
   Test NaN: NAN is FP_NAN, MathIsValidNumber(NaN)=false
   Test Inf: INF is FP_INFINITE, MathIsValidNumber(inf)=false
   Test Normal: 1.2345E+06 is FP_NORMAL, MathIsValidNumber(normal)=true
   Test Subnormal: 1.11254E-308 is FP_SUBNORMAL, MathIsValidNumber(sub_normal)=true
   Test Zero: -0 is FP_ZERO, MathIsValidNumber(zero)=true
*/
//+------------------------------------------------------------------+
```

另见

真实型（双精度型，浮点型）, MathIsValidNumber

© 2000-2025, MetaQuotes Ltd.

# MathCeil

函数从上返回最接近的整数值。

```
double  MathCeil(
   double  val      // 数字
   );
```

参量

*val*

[in] 数字值

返回值

数字值代表最少总数超出或者等于val。

注释

取代MathCeil()函数，可以使用 ceil()。

示例：

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明转换变量
   double value=0;                     // 用于MathCeil转换的实数
   int    ceil_value=0;                // 在这里获得结果
//--- 按实数的十进制增量数进行循环
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- 增加数值，
      //--- 从上面获取最接近的整数值
      //--- 并在日志中显示控件值
      value+=0.1;
      ceil_value=(int)MathCeil(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, ceil value: %d",value,ceil_value);
      /*
      result:
      value: 0.1, ceil value: 1
      value: 0.2, ceil value: 1
      value: 0.3, ceil value: 1
      value: 0.4, ceil value: 1
      value: 0.5, ceil value: 1
      value: 0.6, ceil value: 1
```

© 2000-2025, MetaQuotes Ltd.

```
      value: 0.7, ceil value: 1
      value: 0.8, ceil value: 1
      value: 0.9, ceil value: 1
      value: 1.0, ceil value: 1
      value: 1.1, ceil value: 2
      value: 1.2, ceil value: 2
      value: 1.3, ceil value: 2
      value: 1.4, ceil value: 2
      value: 1.5, ceil value: 2
      value: 1.6, ceil value: 2
      value: 1.7, ceil value: 2
      value: 1.8, ceil value: 2
      value: 1.9, ceil value: 2
      value: 2.0, ceil value: 2
      value: 2.1, ceil value: 3
      value: 2.2, ceil value: 3
      value: 2.3, ceil value: 3
      value: 2.4, ceil value: 3
      value: 2.5, ceil value: 3
      value: 2.6, ceil value: 3
      value: 2.7, ceil value: 3
      value: 2.8, ceil value: 3
      value: 2.9, ceil value: 3
      value: 3.0, ceil value: 3
      value: 3.1, ceil value: 4
      */
   }
}
```

© 2000-2025, MetaQuotes Ltd.

# MathCos

函数返回余弦角。

```
double  MathCos(
   double  value      // 数字
   );
```

参量

*值*

[in] 角度

返回值

双精度值在-1到1之间。

注释

取代MathCos()函数可以使用 cos()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的余弦值
   vector Y=MathCos(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

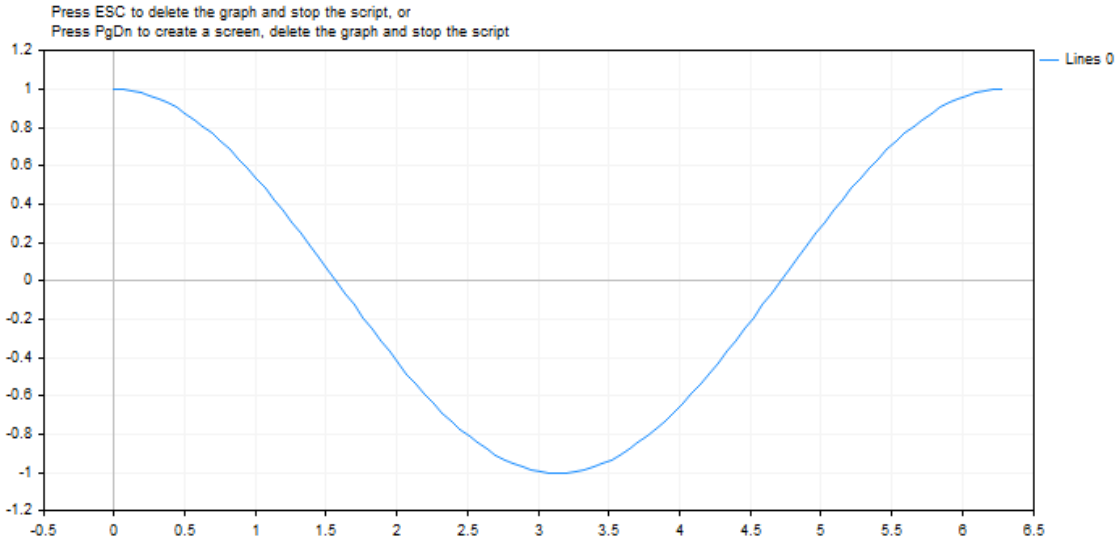
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathExp

函数返回e到d的值。

```
double  MathExp(
   double  value      // 数字e的功能
   );
```

参量

*值*

[in] 指定数字功率

返回值

许多双精度函数，函数返回INF（无限大），在缺少MathExp时返回0

注释

取代MathExp()可以使用 exp().

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到8的9个值，步长为1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- 计算“e”欧拉数)的每个X向量值的幂数
   X=MathExp(X);
   Print("MathExp(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形（截取屏幕截图）并退出
   while(!IsStopped())
```

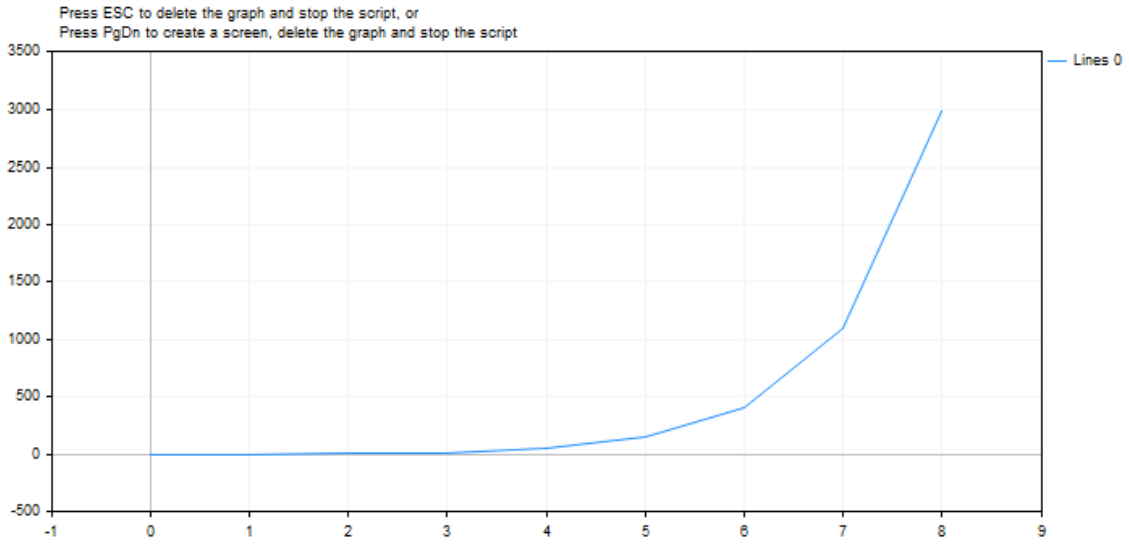
© 2000-2025, MetaQuotes Ltd.

```
      {
       if(StopKeyPressed())
          break;
       Sleep(16);
      }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRIT
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathFloor

函数返回下面最接近的整数值。

```
double  MathFloor(
   double  val     // 数字
   );
```

参量

*val*

[in] 数值。

返回值

数字值代表最大整数少于或者等于val。

注释

取代MathFloor()可以使用 floor()。

示例：

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明转换变量
   double value=0;                     //用于MathFloor转换的实数
   int    floor_value=0;               // 在这里获得结果
//--- 按实数的十进制增量数进行循环
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- 增加数值，
      //--- 从下面获取最接近的整数值
      //--- 并在日志中显示控件值
      value+=0.1;
      floor_value=(int)MathFloor(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, floor value: %d",value,floor_value);
      /*
      result:
      value: 0.1, floor value: 0
      value: 0.2, floor value: 0
      value: 0.3, floor value: 0
      value: 0.4, floor value: 0
      value: 0.5, floor value: 0
      value: 0.6, floor value: 0
```

© 2000-2025, MetaQuotes Ltd.

```
      value: 0.7, floor value: 0
      value: 0.8, floor value: 0
      value: 0.9, floor value: 0
      value: 1.0, floor value: 1
      value: 1.1, floor value: 1
      value: 1.2, floor value: 1
      value: 1.3, floor value: 1
      value: 1.4, floor value: 1
      value: 1.5, floor value: 1
      value: 1.6, floor value: 1
      value: 1.7, floor value: 1
      value: 1.8, floor value: 1
      value: 1.9, floor value: 1
      value: 2.0, floor value: 2
      value: 2.1, floor value: 2
      value: 2.2, floor value: 2
      value: 2.3, floor value: 2
      value: 2.4, floor value: 2
      value: 2.5, floor value: 2
      value: 2.6, floor value: 2
      value: 2.7, floor value: 2
      value: 2.8, floor value: 2
      value: 2.9, floor value: 2
      value: 3.0, floor value: 3
      value: 3.1, floor value: 3
      */
     }
```

© 2000-2025, MetaQuotes Ltd.

# MathLog

函数返回一组自然对数。

```
double  MathLog(
   double  val      // 取对数的数值
   );
```

参量

*val*

[in] 可以找到对数值。

返回值

如果成功出现自然对数val，如果val是负的，函数返回NaN(未定值)。如果val等于0，函数返回INF (无限)。

注释

取代MathLog() 可以使用 log()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到8的9个值，步长为1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- 计算X向量每个值的对数
   X=MathLog(X);
   Print("MathLog(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

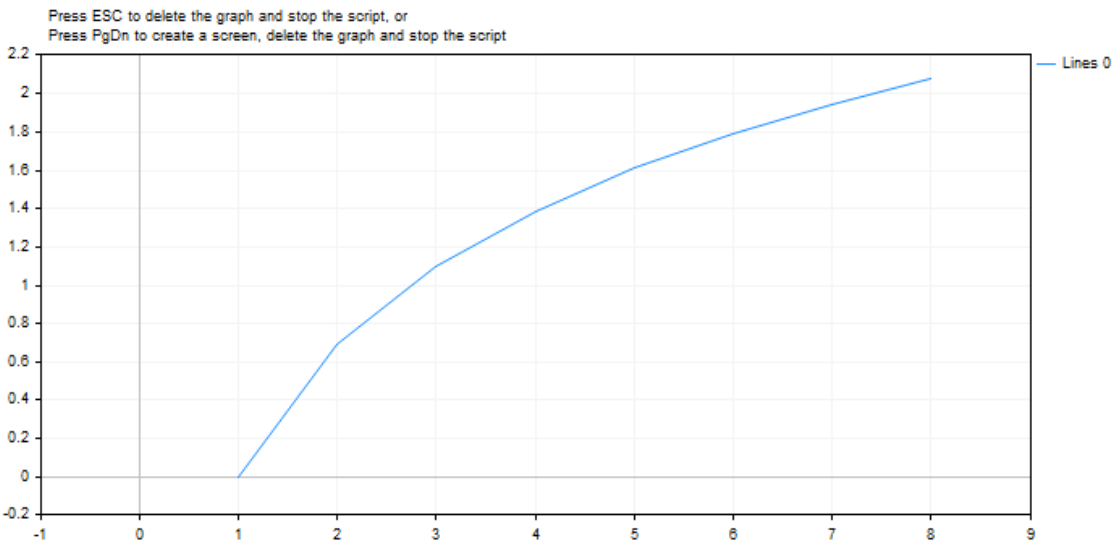
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8]
   MathLog(X) =
   [-inf,0,0.6931471805599453,1.09861228866811,1.386294361119891,1.6094379124341,1.791
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathLog

以10为底返回对数。

```
double  MathLog10(
   double  val      // 取对数的数值
   );
```

参量

*val*

[in] 数字值是计算的常用对数。

返回值

成功了是常用对数，如果val是负的，函数返回NaN,(待定值)，如果val等于0，函数返回INF (无限大)

注释

取代MathLog10()可以使用 log10()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到8的9个值，步长为1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- 计算X向量每个值的十进制对数
   X=MathLog10(X);
   Print("MathLog10(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

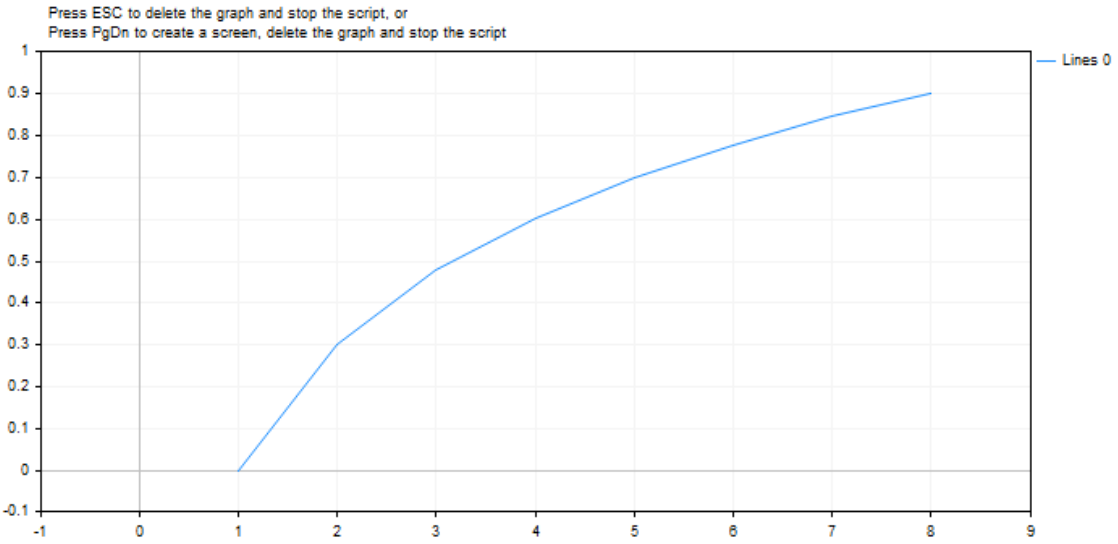
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8]
   MathLog10(X) =
   [-inf,0,0.3010299956639812,0.4771212547196624,0.6020599913279624,0.6989700043360189
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathMax

函数返回最大的量个值。

```
double  MathMax(
   double  value1,      // 第一值
   double  value2       // 第二值
   );
```

参量

*value1*

[in] 第一个数字值。

*value2*

[in] 第二个数字值。

返回值

两个值中的最大值。

注释

取代MathMax()可以使用 fmax()。函数 fmax(), fmin(), MathMax(), MathMin() 是整型的，而非铸字的双精度类型。

如果不同类型参量通过函数传递，副类型参量自动转到主类型。该类型返回值相当于主类型返回值。

如果相同数据类型传递，不用执行铸字。

示例：

```
//--- 输入参数
input int                  InpPeriod = 10;            // 移动平均线计算周期
input ENUM_MA_METHOD       InpMethod = MODE_SMA;      // 移动平均线计算方法
input ENUM_APPLIED_PRICE   InpPrice  = PRICE_CLOSE;   // 移动平均线计算价格

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 如果移动平均线周期设置为小于1的值，则使用默认值(10)
   int period=(InpPeriod<1 ? 10 : InpPeriod);
//--- 创建移动平均线指标句柄
   int handle=iMA(Symbol(),Period(),period,0,InpMethod,InpPrice);
   if(handle==INVALID_HANDLE)
     {
      Print("Failed to create the Moving Average indicator handle. Error ",GetLastErro
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 获得当前卖价
   double bid=0;
   ResetLastError();
   if(!SymbolInfoDouble(Symbol(),SYMBOL_BID,bid))
     {
      Print("Failed to get Bid price. Error ",GetLastError());
      return;
     }
//--- 获得当前柱形图的移动平均线值
   double array[1];
   int    copied=CopyBuffer(handle,0,0,1,array);
   if(copied!=1)
     {
      Print("Failed to get Moving Average data. Error ",GetLastError());
      return;
     }
//--- 获取两者中最高的价格 卖价和移动平均线值)，并在日志中显示结果数据
   double max_price=MathMax(bid,array[0]);
   PrintFormat("Bid: %.*f, Moving Average: %.*f, highest price of the two: %.*f",_Digi
   PrintFormat("Bid price %s moving average",(bid>array[0] ? "higher" : bid<array[0] ?
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathMin

函数返回两个值中的最小值。

```
double  MathMin(
   double  value1,      // 第一值
   double  value2       // 第二值
   );
```

参量

*value1*

[in] 第一数字值。

*value2*

[in] 第二数字值

返回值

两值中最小的。

注释

取代MathMin()可以使用 fmin()。函数 fmax()， fmin(), MathMax(), MathMin() 是整型，而非铸字的双精度类型。

如果是不同类型参量传递到函数中，副类型参量自动 转 到主类型。返回值类型与主类型相同。

如果数据类型传递，不需要执行铸字。

示例：

```
//--- 输入参数
input uint  InpBars  =  100000;  // Desired number of bars

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获得服务器上可用柱形图的数量
   uint bars_total = Bars(Symbol(),Period());
   if(bars_total==0)
     {
      PrintFormat("Data for timeseries %s %s not yet generated in the terminal. Please
      return;
     }
//--- 从两个值中获取最小柱形图数量 - 从服务器上可用的值和从设置中请求的值
   int bars = (int)MathMin(bars_total,InpBars);
//--- 如果请求的柱形图数量多于服务器上可用的柱形图，请在日志中报告此情况
   if(bars_total<InpBars)
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("Number of bars on the server (%u) is less than requested (%u)",bars
//--- 在日志中显示可用于工作的柱形图数量
   Print("Bars available for work: ",bars);
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathMod

函数返回两个数字相除的余数。

```
double  MathMod(
   double  value,       // 股息值
   double  value2       // 除数值
   );
```

参量

*值*

[in] 被除数值。

*value2*

[in] 除数值。

返回值

MathMod函数计算真值 f of val/ y such that val= i * y + f, f是整数，f有与va相同的符号，f的绝对值小于y的绝对值。

注释

取代MathMod()可以使用 fmod()。

示例：

```
#property script_show_inputs

//--- 输入参数
input double   InpDividentValue  =  10;   // 股息值
input double   InpDivisorValue   =  3;    // 除数值

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取输入中输入的数字相除的实数余数
   double res=MathMod(InpDividentValue,InpDivisorValue);
//--- 在日志中记录结果
   PrintFormat("Real remainder when dividing %.2f by %.2f = %.2f",InpDividentValue,Inp
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathPow

函数增加基数的整个功率。

```
double  MathPow(
   double  base,          // 基数
   double  exponent       // 指数值
   );
```

参量

*base*

[in] 基数。

*exponent*

[in] 指数值

返回值

基数值提升到特定功率。

注释

取代MathPow()可以使用 pow()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#property script_show_inputs

#include <Graphics\Graphic.mqh>

//--- 输入参数
input double   InpExponentValue  =  2;     // exponent value

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到10的11个值，步长为1
   vector X(11,VectorArange);
   Print("vector X = \n",X);
//--- 计算每个提升到InpExponentValue幂的X向量值
   X=MathPow(X,InpExponentValue);
   Print("MathPow(X,",(string)InpExponentValue,") = \n",X);
```

© 2000-2025, MetaQuotes Ltd.

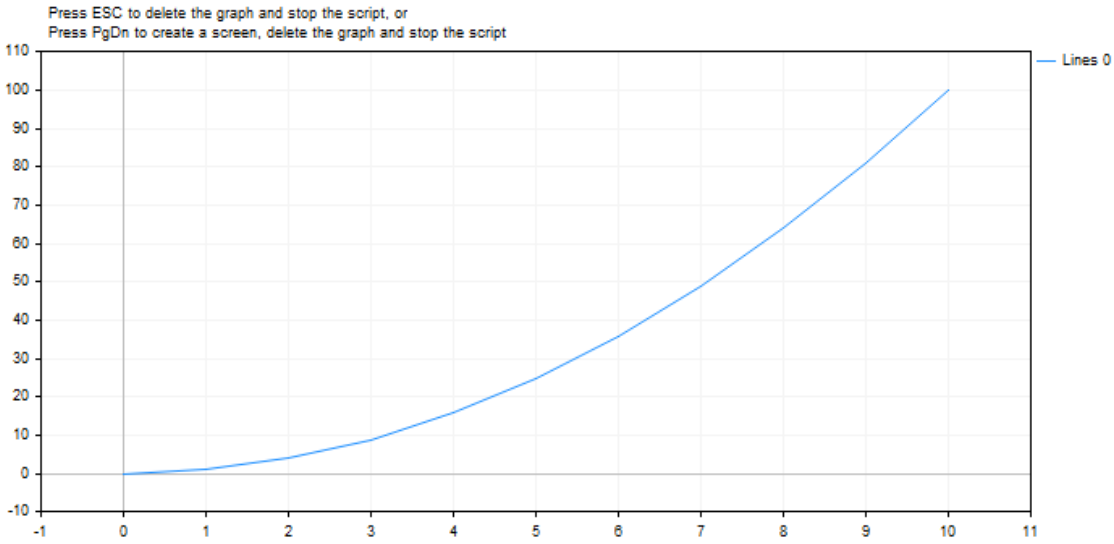
```
//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8,9,10]
   MathPow(X,2.0) =
   [0,1,4,9,16,25,36,49,64,81,100]
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：

© 2000-2025, MetaQuotes Ltd.




© 2000-2025, MetaQuotes Ltd.

# MathRand

在0到32767之间返回随机整数。

```
int  MathRand();
```

返回值

整数值在0到32767之间。

注释

在第一次调用前，调用 MathSrand 建立初始状态伪随机数是有必要的。

注释

取代MathRand()可以使用 rand()。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置新的初始状态，在每次启动时生成一系列伪随机整数
   MathSrand(GetTickCount());
//--- 在循环中，在日志中显示10个生成的伪随机整数
   for(int i=0; i<10; i++)
     {
      int rand_value=MathRand();
      PrintFormat("Pseudorandom integer №%d: %u",i+1,rand_value);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathRound

函数四舍五入返回最接近的整数值。

```
double  MathRound(
   double  value      // 全值
   );
```

参量

*值*

[in] 四舍五入前的数值。

返回值

四舍五入到最接近的整数值。

注释

取代MathRound()可以使用 round().

示例：

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明转换变量
   double value=0;                     // 用于MathRound转换的实数
   int    round_value=0;               // 在这里获得结果
//--- 按实数的十进制增量数进行循环
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- 增加实数值，
      //--- 获得四舍五入到最接近的整数值。
      //--- 并在日志中显示控件值
      value+=0.1;
      round_value=(int)MathRound(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, round value: %d",value,round_value);
      /*
      result:
      value: 0.1, round value: 0
      value: 0.2, round value: 0
      value: 0.3, round value: 0
      value: 0.4, round value: 0
      value: 0.5, round value: 1
      value: 0.6, round value: 1
```

© 2000-2025, MetaQuotes Ltd.

```
      value: 0.7, round value: 1
      value: 0.8, round value: 1
      value: 0.9, round value: 1
      value: 1.0, round value: 1
      value: 1.1, round value: 1
      value: 1.2, round value: 1
      value: 1.3, round value: 1
      value: 1.4, round value: 1
      value: 1.5, round value: 2
      value: 1.6, round value: 2
      value: 1.7, round value: 2
      value: 1.8, round value: 2
      value: 1.9, round value: 2
      value: 2.0, round value: 2
      value: 2.1, round value: 2
      value: 2.2, round value: 2
      value: 2.3, round value: 2
      value: 2.4, round value: 2
      value: 2.5, round value: 3
      value: 2.6, round value: 3
      value: 2.7, round value: 3
      value: 2.8, round value: 3
      value: 2.9, round value: 3
      value: 3.0, round value: 3
      value: 3.1, round value: 3
      */
   }
}
```

© 2000-2025, MetaQuotes Ltd.

# MathSin

返回特定角的正弦。

```
double  MathSin(
   double  value      // 弧度参数
   );
```

参量

*值*

[in] 角的弧度。

返回值

测量角的正弦，返回在-1到1之间的值。

注释

取代MathSin()可以使用 sin()。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的正弦值
   vector Y=MathSin(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
```

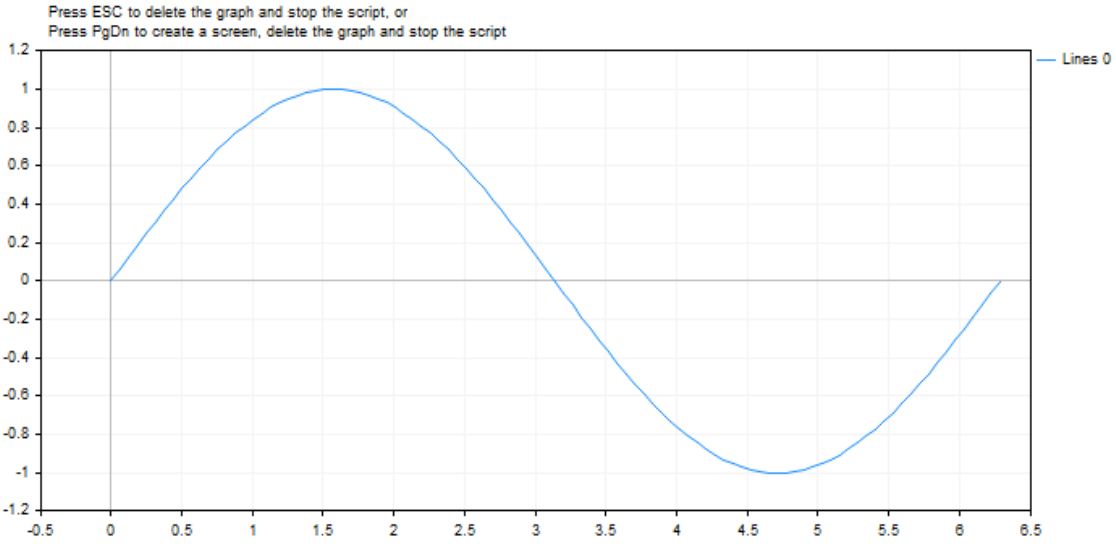
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathSqrt

返回数字的平方根。

```
double  MathSqrt(
   double  value      // 正值
   );
```

参量

*值*

[in] 正值

返回值

平方根值，值是负的，MathSqrt返回NaN(待定值)。

注释

取代MathSqrt()可以使用 sqrt().

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到10的11个值，步长为1
   vector X(11,VectorArange);
   Print("vector X = \n",X);
//--- 计算X向量每个值的平方根
   X=MathSqrt(X);
   Print("MathSqrt(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
```

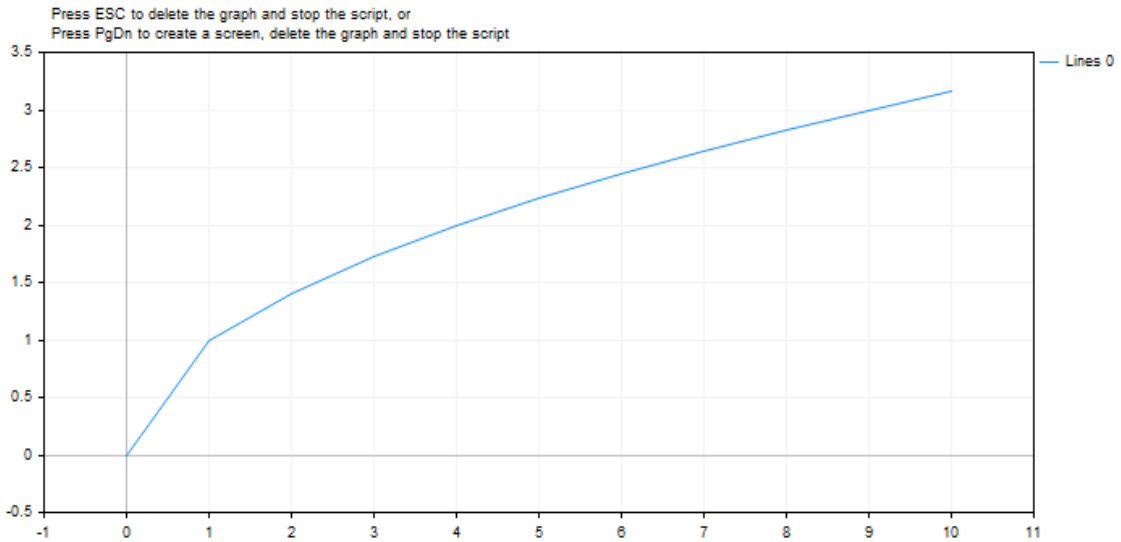
© 2000-2025, MetaQuotes Ltd.

```
      {
       if(StopKeyPressed())
          break;
       Sleep(16);
      }

//--- 清除
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8,9,10]
   MathSqrt(X) =
   [0,1,1.414213562373095,1.732050807568877,2,2.23606797749979,2.449489742783178,2.645
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRIT
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

© 2000-2025, MetaQuotes Ltd.

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathSrand

建立伪随机整数产生序列出发点。

```
void  MathSrand(
   int  seed      // 初始化数字
   );
```

参量

*seed*

[in] 随机数开始数字的排列。

返回值

没有返回值。

注意

MathRand() 函数用于生成一系列的伪随机数。通过正在初始化的数字调用MathSrand() 允许始终生产同系列的伪随机数。

要确定收到非循环数列，使用调用MathSrand(GetTickCount())，由于GetTickCount() 值从操作系统开始时增加并且在49天内不重复，一直到超出内置毫秒计数器。使用MathSrand(TimeCurrent()) 不合适，因为TimeCurrent() 函数返回长时间未更改的最近订单号的时间，例如周末。

使用MathSrand()函数为指标和EA而初始化的随机数生成程序在 OnInit()处理程序被更好的执行，它可以有助于您防止以下在OnTick() 和 OnCalculate()多次重启生成程序。

您可以使用srand()函数来替代MathSrand()函数。

例如：

```
#property description "The indicator shows the central limit theorem, which states:"
#property description "The sum of a sufficiently large number of weakly dependent rand
#property description "having approximately equal magnitude (none of the summands dom
#property description "or makes a determining contribution to the sum), has a distribu

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 图形结构属性
#property indicator_label1  "Label"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRoyalBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  5
//--- 输入变量
input int       sample_number=10;
//--- 用于绘制分布图的指标缓冲区
double          LabelBuffer[];
//--- 订单号计数器
double          ticks_counter;
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- 从现在到过去转移指标缓冲区
   ArraySetAsSeries(LabelBuffer,true);
//--- 初始化随机数生成程序
   MathSrand(GetTickCount());
//--- 初始化订单号计数器
   ticks_counter=0;
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 为零值计数器重置指标缓冲区
   if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- 增加计数器
   ticks_counter++;
//--- 我们应该周期性的重置计数器，恢复分配
   if(ticks_counter>100)
     {
      Print("We've reset the indicator values, let's start filling the cells once agai
      ticks_counter=0;
     }
//--- 得到从0到7三个数字总和的随机值样本
   for(int i=0;i<sample_number;i++)
     {
      //--- 计算单元标引，随机数下跌三个数字的总和
      int rand_index=0;
      //--- 得到从0到7的三个随机数字
      for(int k=0;k<3;k++)
        {
         //--- 除以7的余数将返回从0到6的值
         rand_index+=MathRand()%7;
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 增加rand_index单元数的值为1
      LabelBuffer[rand_index]++;
     }
//--- 退出OnCalculate() 处理程序
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathTan

函数返回正切数。

```
double  MathTan(
   double  rad      // 弧度参数
   );
```

参量

*rad*

[in] 弧度角

返回值

rad正切，如果rad大于或等于263，或者小于或等于-263，丢掉有意值，在此情况下，函数返回不确定。

注释

取代MathTan()可以使用 tan().

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=-263;
//--- 获取从-263到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的正切值
   vector Y=MathTan(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);
```

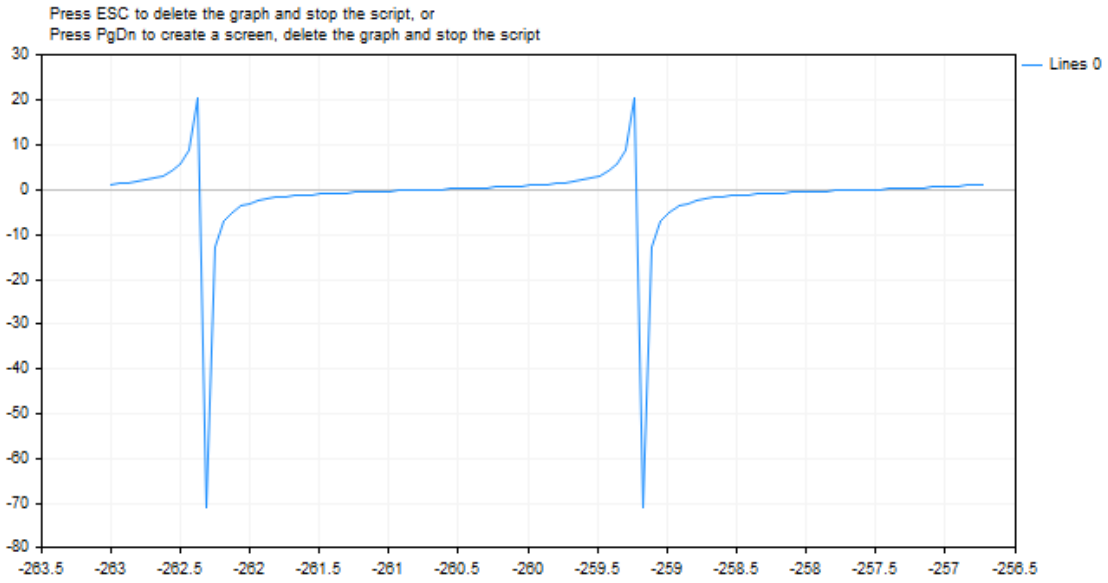
© 2000-2025, MetaQuotes Ltd.

```
//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC,则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn,并且成功抓取图形屏幕截图,则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
```

© 2000-2025, MetaQuotes Ltd.

```
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRIT
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathIsValidNumber

检测实数的正确性。

```
bool  MathIsValidNumber(
   double  number      // 检测数
   );
```

参量

*number*

[in] 检测数值。

返回值

返回真值，如果检测值是可以接受的实数。如果检测值是正无穷或者负无穷大，或者没有数字 (NaN),函数返回错误值。

示例：

```
   double abnormal=MathArcsin(2.0);
   if(!MathIsValidNumber(abnormal)) Print("Attention! MathArcsin(2.0) = ",abnormal);
```

另见

真实型（双精度，浮点）

© 2000-2025, MetaQuotes Ltd.

# MathExpm1

返回MathExp(x)-1表达式值。

```
double  MathExpm1(
   double  value      // 数值e的能力
   );
```

参数

*值*

[in] 指定能力的数值。

返回值

双精度类型值。溢出的情况下，函数返回INF (∞)，下溢的情况下MathExpm1返回0。

注意

x值接近0时，MathExpm1(x)函数会生成精确于MathExp(x)-1函数的值。

您可以使用expm1()函数，代替MathExpm1()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到8的9个值，步长为1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- 计算（"e"欧拉数）的每个X向量值的幂数- 1
   X=MathExpm1(X);
   Print("MathExpm1(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);
```

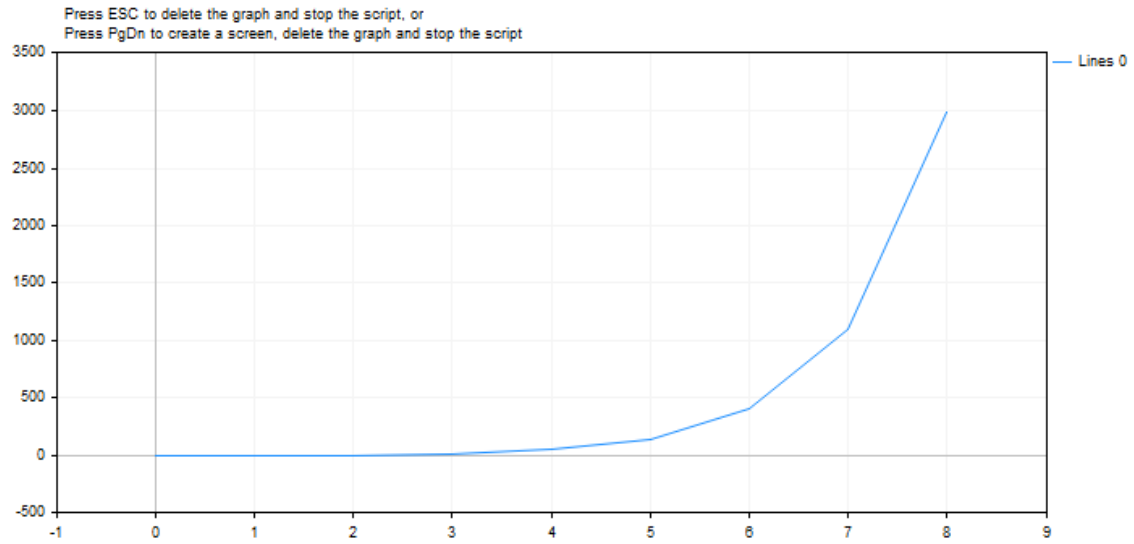
© 2000-2025, MetaQuotes Ltd.

```
//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
```

© 2000-2025, MetaQuotes Ltd.

```
    ExtGraph.Update();
   }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
   {
    string file_names[];
    ResetLastError();
    int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
    if(selected<1)
      {
       if(selected<0)
          PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
       return false;
      }

    bool res=false;
    if(ChartSetInteger(0,CHART_SHOW,false))
       res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
    ChartSetInteger(0,CHART_SHOW,true);
    return(res);
   }
```

结果：




另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# MathLog1p

返回MathLog(1+x)表达式值。

```
double  MathLog1p(
   double  value      // 对数值
   );
```

参数

*值*

[in] 将被计算的对数值。

返回值

如果成功，返回值（值+1）的自然对数。如果值小于 -1，则函数返回NaN（未定义值）。如果值等于-1，函数返回INF（∞）。

注意

x值接近0时，MathLog1p(x)函数会生成精确于MathLog(1+x)函数的值。

您可以使用log1p()函数，代替MathLog1p()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取从0到8的9个值，步长为1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- 对于每个向量值，计算表达式的对数（1 + X向量值）
   X=MathLog1p(X);
   Print("MathLog1p(X) = \n",X);

//--- 将计算值从向量传输到数组
   double y_array[];
   X.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(y_array,clrDodgerBlue);
```

© 2000-2025, MetaQuotes Ltd.

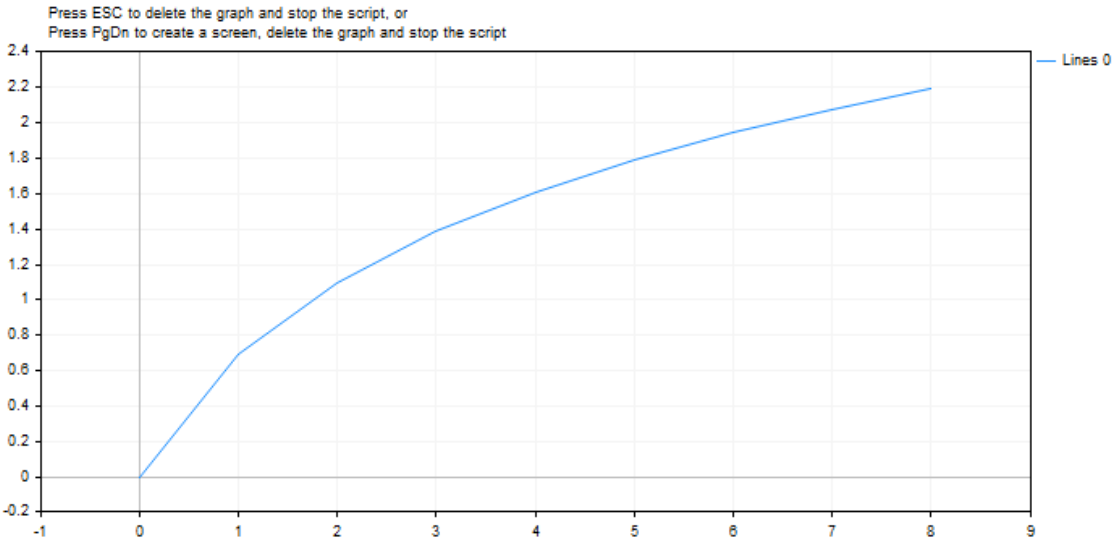
```
//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8]
   MathLog1p(X) =
   [0,0.6931471805599453,1.09861228866811,1.386294361119891,1.6094379124341,1.7917594
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：

© 2000-2025, MetaQuotes Ltd.




另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# MathArccosh

返回双曲反余弦。

```
double  MathArccosh(
   double  value     // 1 <= 值< ∞
   );
```

参数

*值*

[in] 将被计算双曲反余弦的值。

返回值

数值的双曲反余弦。如果该值小于+1，则该函数返回NaN（未定义值）。

注意

您可以使用acosh()函数，代替MathArccosh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/10);
   delta[0]=1;
//--- 获取从-1到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲反余弦值
   vector Y=MathArccosh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形（截取屏幕截图）并退出
```

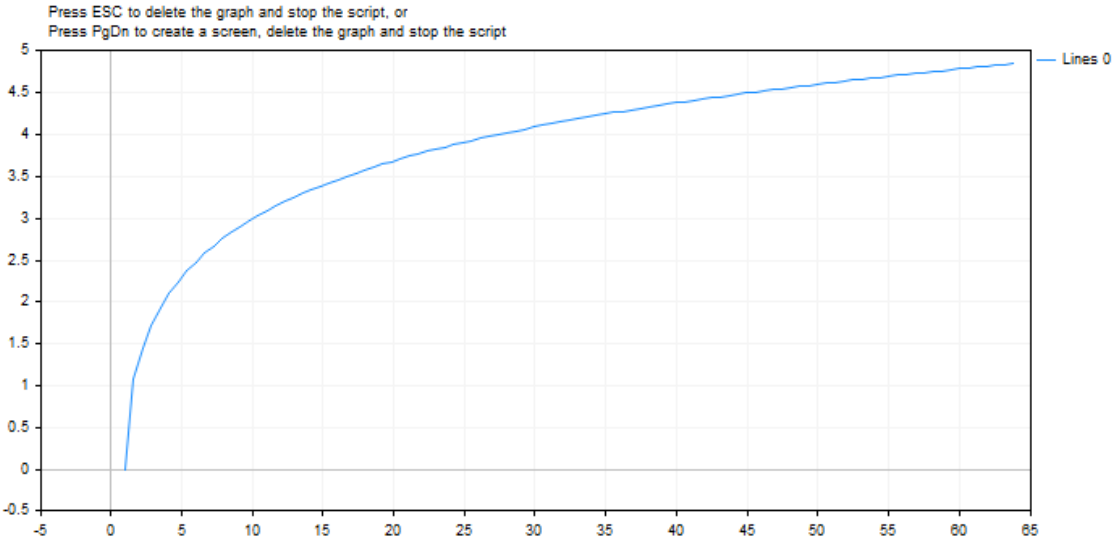
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and saves the image to a file                  |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# MathArcsinh

返回双曲反正弦。

```
double  MathArcsinh(
   double  value      // -∞ < 值 < +∞
   );
```

参数

*val*

[in] 将被计算双曲反正弦的值。

返回值

数值的双曲反正弦。

注意

您可以使用asinh()函数，代替MathArcsinh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/10);
   delta[0]=-31;
//--- 获取从-31到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲反正弦值
   vector Y=MathArcsinh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
```

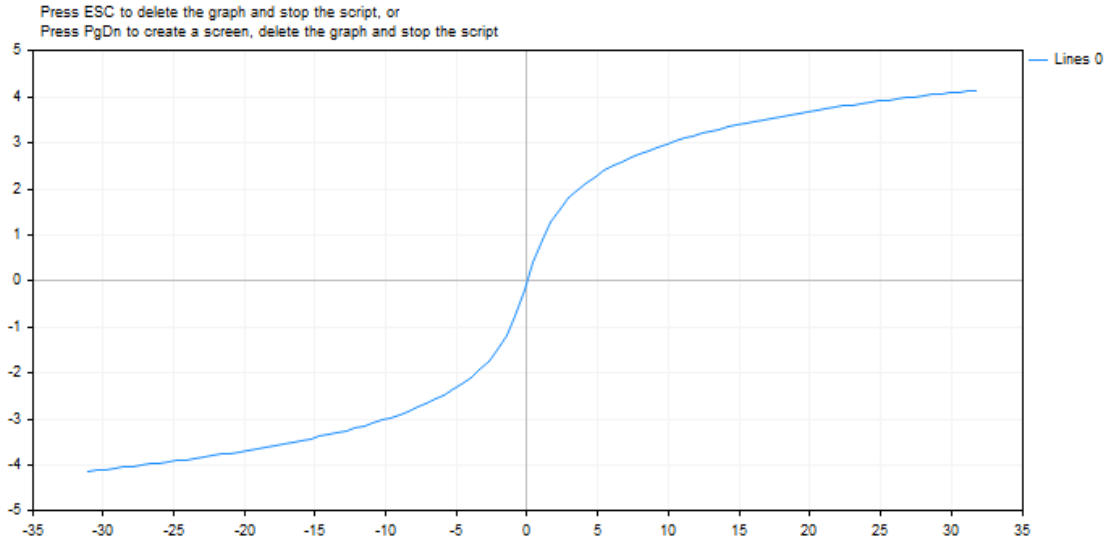
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# MathArctanh

返回双曲反正切。

```
double  MathArctanh(
   double  value      // 值的范围 -1 < 值 < 1
   );
```

参数

*值*

[in] 代表正切的 -1 < 值 < 1 范围的数值。

返回值

数值的双曲反正切。

注意

您可以使用atanh()函数，代替MathArctanh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(34,2*M_PI/100);
   delta[0]=-1;
//--- 获取从-1到2pi的34个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲反正切值
   vector Y=MathArctanh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
```

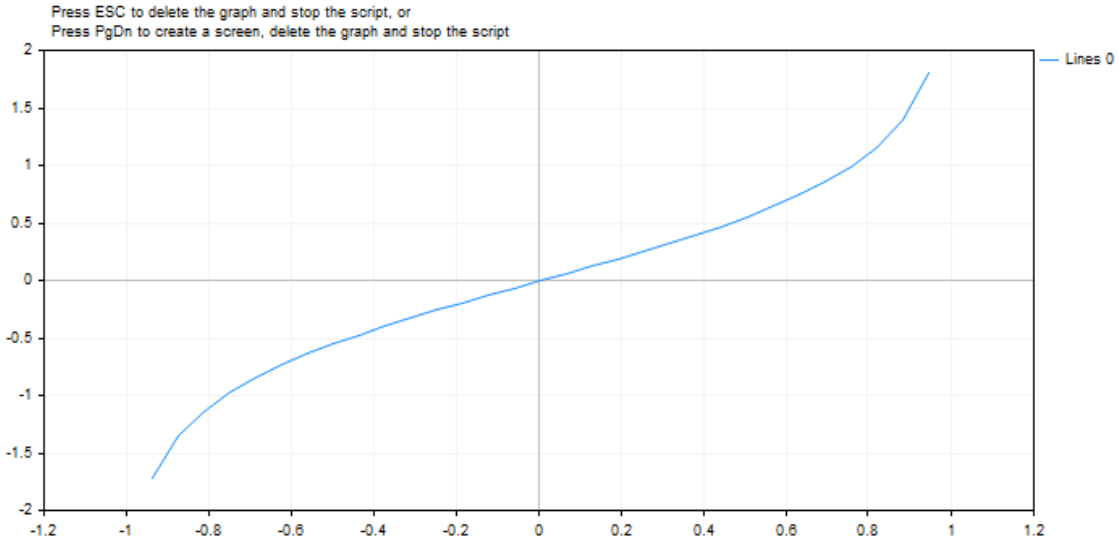
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathCosh

返回数值的双曲余弦。

```
double  MathCosh(
   double  value      // 数值
   );
```

参数

*值*

[in] 值。

返回值

数值的双曲余弦，其值范围在+1到+∞。

注意

您可以使用cosh()函数，代替MathCosh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲余弦值
   vector Y=MathCosh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

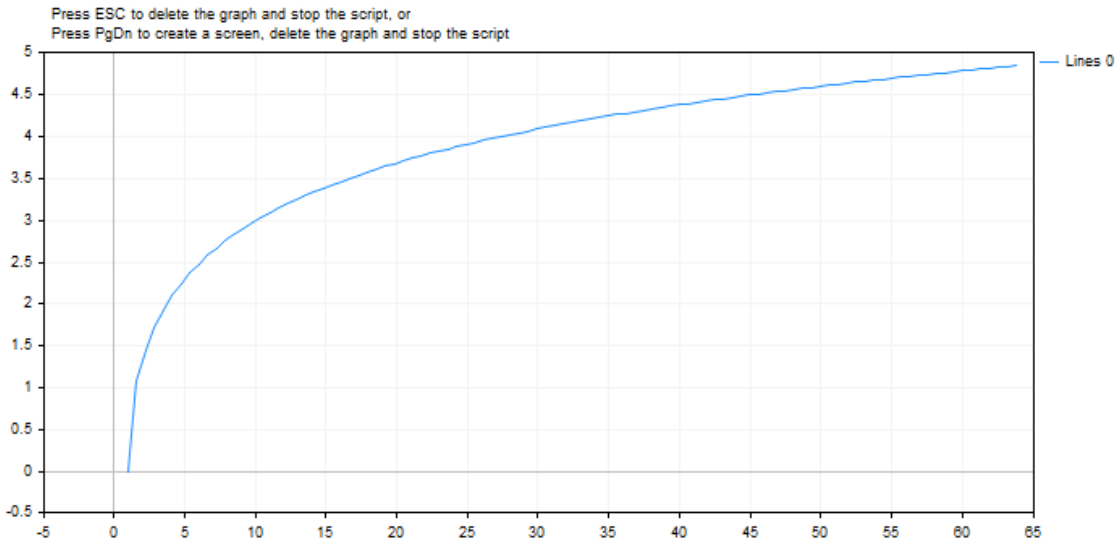
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathSinh

返回数值的双曲正弦。

```
double  MathSinh(
   double  value      // 数值
   );
```

参数

*值*

[in] 值。

返回值

数值的双曲正弦。

注意

您可以使用sinh()函数，代替MathSinh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲正弦值
   vector Y=MathSinh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 (截取屏幕截图)并退出
```

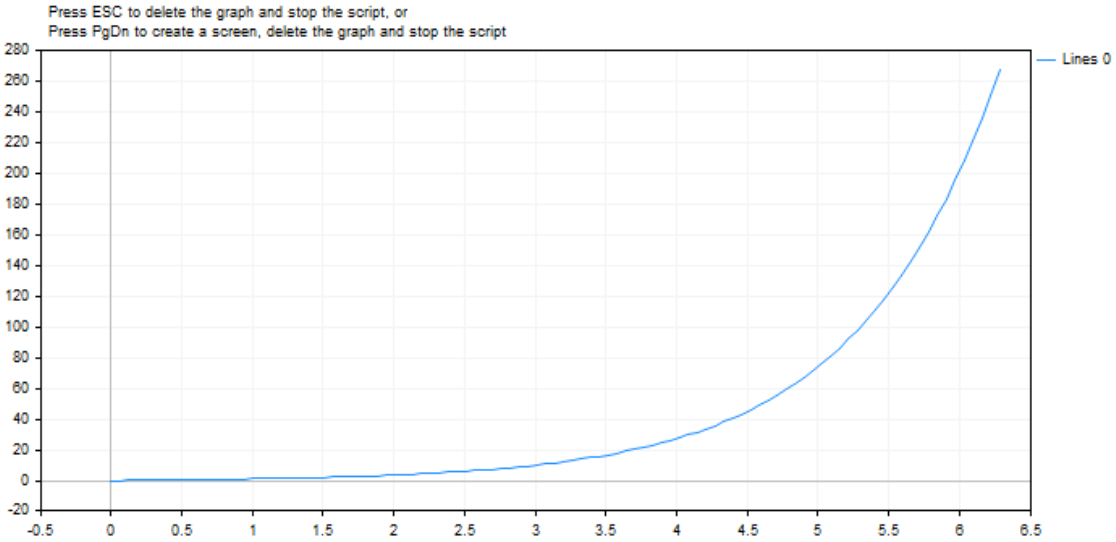
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




© 2000-2025, MetaQuotes Ltd.

# MathTanh

返回数值的双曲正切。

```
double  MathTanh(
   double  value      // 数值
   );
```

参数

*值*

[in] 值。

返回值

数值的双曲正切，其值范围在 -1 到 +1。

注意

您可以使用tanh()函数，代替MathTanh()函数。

示例：

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- 获取从0到2pi的101个值，步长为delta
   vector X=delta.CumSum();
//--- 计算X向量每个值的双曲正切值
   vector Y=MathTanh(X);

//--- 将计算值从向量转移到数组
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- 绘制计算出的向量值的图形
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- 等待按下Escape或PgDn键删除图形 截取屏幕截图)并退出
```

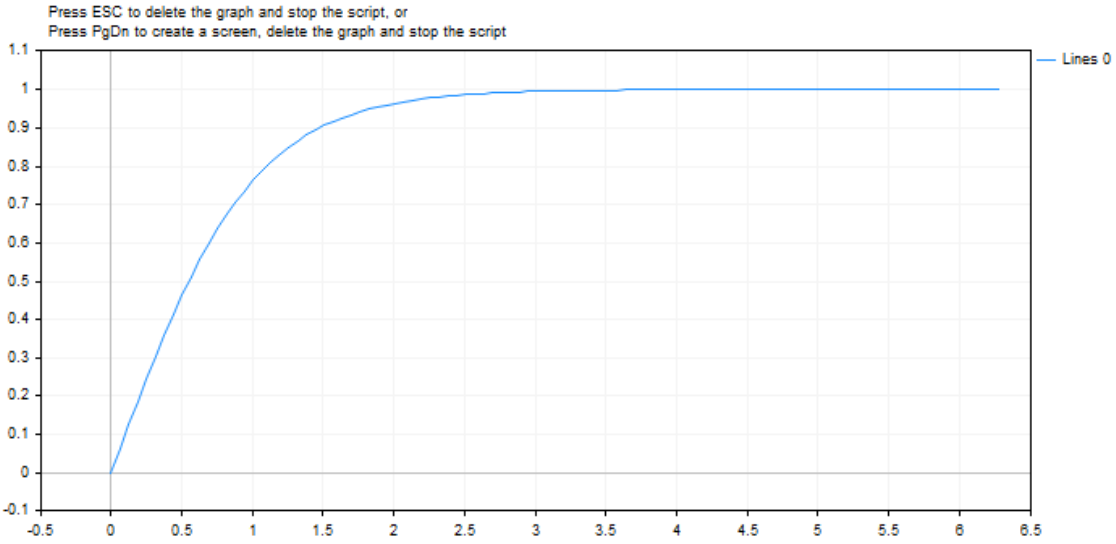
© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- 清除
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- 如果按下ESC，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- 果按下PgDn，并且成功抓取图形屏幕截图，则返回'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- 返回'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
```

© 2000-2025, MetaQuotes Ltd.

```
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

结果：




另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# MathSwap

在ushort类型值中改变字节的顺序

```
ushort  MathSwap(
   ushort  value      //值
   );
```

参数

*值*

[in] 用于改变字节顺序的值。

返回值

具有相反字节顺序的ushort值。

# MathSwap

在uint类型值中改变字节的顺序。

```
uint  MathSwap(
   uint   value      // 值
   );
```

参数

*值*

[in] 用于改变字节顺序的值。

返回值

具有相反字节顺序的uint值。

# MathSwap

在ulong类型值中改变字节的顺序。

```
ulong  MathSwap(
   ulong  value      //值
   );
```

参数

*值*

[in] 用于改变字节顺序的值。

返回值

具有相反字节顺序的ulong值。

示例：

© 2000-2025, MetaQuotes Ltd.

```
#property script_show_inputs

input ulong  InpLongValue  =  1;     // 在这里输入任何ulong值
input uint   InpIntValue   =  2;     // 在这里输入任何uint值
input ushort InpShortValue =  3;     // 在这里输入任何ushort值

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {

//--- 在日志中以十进制和二进制形式记录MathSwap()输入和转换的值
   Print(ValueDescription(InpLongValue));
   Print(ValueDescription(InpIntValue));
   Print(ValueDescription(InpShortValue));
   /*
   result:
   ulong value: 1
   ulong value: 72057594037927936 using MathSwap()
   binary ulong value: 0000000000000000000000000000000000000000000000000000000000000001
   binary ulong value: 0000000100000000000000000000000000000000000000000000000000000000

   uint value: 2
   uint value: 33554432 using MathSwap()
   binary uint value: 00000000000000000000000000000010
   binary uint value: 00000010000000000000000000000000 using MathSwap()

   ushort value: 3
   ushort value: 768 using MathSwap()
   binary ushort value: 0000000000000011
   binary ushort value: 0000001100000000 using MathSwap()
   */
  }
//+------------------------------------------------------------------+
//| Return the text describing the variable values                   |
//+------------------------------------------------------------------+
template <typename T>
string ValueDescription(T x)
  {
   int    num_bits   = sizeof(T)*8;
   string type_name  = typename(T);
   string bin_x      = NumberToBinaryString(x);
   string bin_swap_x = NumberToBinaryString(MathSwap(x));
   return(StringFormat("%s value: %lld\n%s value: %lld using MathSwap()\nbinary %s va
  }
//+------------------------------------------------------------------+
//| Return the binary representation of a number as a string         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
template <typename T>
string NumberToBinaryString(T x)
  {
   string res  = "";
   int    i    = -1;
   uchar  size = sizeof(T)*8-1;
   ulong  mask = (ulong)1<<size;
   while(!((x<<++i)& mask));
   for(; i <=size; i++)
      res += !((x<<i)& mask) ? "0" : "1";
   return res;
  }
```

另见

网络函数、SocketRead、SocketSend、SocketTlsRead、SocketTlsReadAvailable、SocketTlsSend

© 2000-2025, MetaQuotes Ltd.

# 字符串函数

用来与 字符串 型数据一起执行的函数。

| 函数 | 功能 |
| --- | --- |
| StringAdd | 在给出子连接上添加字符串 |
| StringBufferLen | 为字符串返回分配式缓冲器的大小 |
| StringCompare | 比较两个字符串并且如果第一个字符串大于第二个返回1；0-如果两个字符串相等；-1（负一）-如果第一个字符串小于第二个 |
| StringConcatenate | 实现一串参数传递 |
| StringFill | 通过挑选出来的交易品种填满指定字符串 |
| StringFind | 在字符串里搜索子串 |
| StringGetCharacter | 在规定字符串位置返回数字值 |
| StringInit | 通过指定交易品种初始化字符串并提供指定字符串长度 |
| StringLen | 在字符串里返回交易品种数字 |
| StringSetLength | 设置一个字符串的指定长度（以字符为单位） |
| StringReplace | 通过交易品种集合序列替代字符串发现的所有子字符串 |
| StringReserve | 保留内存中字符串的指定大小缓冲区 |
| StringSetCharacter | 在指定位置返回复制后的交易品种的改变值 |
| StringSplit | 在指定字符串通过指定分隔符得到子字符串，返回所得子字符串的数量 |
| StringSubstr | 从指定位置提取文本串 |
| StringToLower | 通过存储单位把所有交易品种中已选的字符串转为小写 |
| StringToUpper | 通过存储单位把所有交易品种中已选的字符串转为资本 |
| StringTrimLeft | 在字符串的左边切断线路供应字符，空间和标号 |
| StringTrimRight | 在字符串的右边切断线路供应字符，空间和标号 |

© 2000-2025, MetaQuotes Ltd.

# StringAdd

在字符串末尾添加子串。

```
bool  StringAdd(
   string&  string_var,          // 将要添加内容的字符串
   string   add_substring        // 被添加的字符串
   );
```

参量

*string_var*

[in][out] 字符串，添加另一组。

*add_substring*

[in] 在源字符串末尾添加ti

返回值

成功返回true,否则是false,为了获得 错误代码，可以调用 GetLastError() 函数。

示例：

```
void OnStart()
  {
   long length=1000000;
   string a="a",b="b",c;
//--- 第一函数
   uint start=GetTickCount(),stop;
   long i;
   for(i=0;i<length;i++)
     {
      c=a+b;
     }
   stop=GetTickCount();
   Print("time for 'c = a + b' = ",(stop-start)," milliseconds, i = ",i);

//--- 第二函数
   start=GetTickCount();
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringAdd(a,b)' = ",(stop-start)," milliseconds, i = ",i);

//--- 第三函数
   start=GetTickCount();
   a="a"; // 重新初始化变量 a
   for(i=0;i<length;i++)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      StringConcatenate(c,a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringConcatenate(c,a,b)' = ",(stop-start)," milliseconds, i = ",i
  }
```

另见

StringConcatenate, StringSplit, StringSubstr

© 2000-2025, MetaQuotes Ltd.

# StringBufferLen

该函数为字符串返回分离式缓冲器的大小。

```
int  StringBufferLen(
   string  string_var      // 字符串
   )
```

参量

*string_var*

[in] 字符串.

返回值

0值表示字符串是恒量，缓冲器大小不改变。-1表示字符串属于客户端，缓冲器内容修正拥有不确定因素。

示例：

```
void OnStart()
  {
   long length=1000;
   string a="a",b="b";
//---
   long i;
   Print("before: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   Print("after: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
  }
```

另见

StringAdd, StringInit, StringLen, StringFill

© 2000-2025, MetaQuotes Ltd.

# StringCompare

该函数是比较两个字符串并且以整数形式返回比较结果。

```
int  StringCompare(
   const string&  string1,                  // 比较的第一个字符串
   const string&  string2,                  // 比较的第二个字符串
   bool           case_sensitive=true       // 为比较选择大小写敏感模式
   );
```

参量

*string1*

[in] 第一个字符串。

*string2*

[in] 第二个字符串。

*case_sensitive=true*

[in] 大小写敏感模式。如果是true，则 "A">"a"。如果是false，则"A"="a"。默认情况下值等于true。

返回值

- -1 (负一), 如果 string1<string2
- 0 (零), 如果 string1=string2
- 1 (一), 如果 string1>string2

注释

字符串是字符与字符相比较，字符根据当前代码页以字母表的顺序比较。

示例：

```
void OnStart()
  {
//--- 哪个更大 - apple or home?
   string s1="Apple";
   string s2="home";

//--- 区分大小写的比较
   int result1=StringCompare(s1,s2);
   if(result1>0) PrintFormat("Case sensitive comparison: %s > %s",s1,s2);
   else
     {
      if(result1<0)PrintFormat("Case sensitive comparison: %s < %s",s1,s2);
      else PrintFormat("Case sensitive comparison: %s = %s",s1,s2);
     }

//--- 不区分大小写的比较
   int result2=StringCompare(s1,s2,false);
   if(result2>0) PrintFormat("Case insensitive comparison: %s > %s",s1,s2);
   else
     {
      if(result2<0)PrintFormat("Case insensitive comparison: %s < %s",s1,s2);
      else PrintFormat("Case insensitive comparison: %s = %s",s1,s2);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
/* 结果:
      区分大小写的比较: Apple < home
      不区分大小写的比较: Apple < home
*/
  }
```

另见

字符串类型, CharToString(), ShortToString(), StringToCharArray(), StringToShortArray(), StringGetCharacter(), 代码页的使用

© 2000-2025, MetaQuotes Ltd.

# StringConcatenate

数据的字串符形式通过并且返回常规字符串大小。参量可以为任意类型。通过参量的总数不得少于2字符或超过64个字符。

```
int  StringConcatenate(
   string&  string_var,    // 要形成的字符串
   void argument1          // 任何简单类型的第一参量
   void argument2          // 任何简单类型的第二参量
   ...                     // 任何简单类型的下一个参量
   );
```

参量

*string_var*

[out] 字符串作为串联结果能够形成。

*argumentN*

[in] 逗号分离值，从2-63任意简单类型参量。

返回值

返回字符串长度，通过参量串联转变成字符串类型而形成，根据与 Print() 和 Comment()相同的规则把参量转变成字符串。与打印不同，结果字符串串联参量并不能被空间分开。

示例：

```
void OnStart()
  {
//--- 声明和定义参与连接的变量
   string text="";
   string text1="This script shows how the StringConcatenate() function works.\n";
   string text2="This is the second line, at the end of which there is a line break co
   string text3="This is line number ";
   int    num3=3;
   string text31=", the number of which is entered into the function as a separate par
   string textN="\n";
   string text4="This is line number 4, preceded by a separate parameter with a line b
   int    length=StringConcatenate(text, text1, text2, text3, num3, text31, textN, tex
   Print(text, "\nLength of the resulting string = ", length);

   /*
  结果
   This script shows how the StringConcatenate() function works.
   This is the second line, at the end of which there is a line break control code.
   This is line number 3, the number of which is entered into the function as a separa
   This is line number 4, preceded by a separate parameter with a line break code.
   Line 5 includes a real number: 0.12345
   Length of the resulting string = 358
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

另 见

StringAdd、StringSplit、StringSubstr

© 2000-2025, MetaQuotes Ltd.

# StringFill

通过指定交易品种填充所选字符串。

```
bool  StringFill(
   string&   string_var,        // 要添加的字符串
   ushort    character          // 添加字符串的交易品种
   );
```

参量

*string_var*

[in][out] 字符串，通过所选交易品种填充。

*character*

[in] 交易品种，字符串填充。

返回值

如果成功返回true，否则是false，获得 错误代码 调用 GetLastError() 。

注释

正确填充字符串表示交易品种直接插入字符串而无需使用新字符串添加或复制的过渡性操作，这样缩短的操作时间。

示例：

```
void OnStart()
  {
   string str;
   StringInit(str,20,'_');
   Print("str = ",str);
   StringFill(str,0);
   Print("str = ",str,": StringBufferLen(str) = ", StringBufferLen(str));
  }
// 结果
//   str = ____________________
//   str =  : StringBufferLen(str) = 20
//
```

另见

StringBufferLen， StringLen， StringInit

© 2000-2025, MetaQuotes Ltd.

# StringFind

在字符串中搜索子字符串。

```
int  StringFind(
   string  string_value,         // 进行搜索的字符串
   string  match_substring,      // 搜索内容
   int     start_pos=0           // 搜索开始位置
   );
```

参量

*string_value*

[in] 字符串，在产生搜索中使用。

*match_substring*

[in] 寻找子串。

*start_pos=0*

[in] 从搜索开始位置索引。

返回值

如果未找到子字串符，从搜索子字串符开始返回字串符中的位置，或是 -1。

示例：

```
#define   RESERVE    100

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取交易品种基础货币和利润货币
   string symbol_currency_base  =SymbolInfoString(Symbol(), SYMBOL_CURRENCY_BASE);
   string symbol_currency_profit=SymbolInfoString(Symbol(), SYMBOL_CURRENCY_PROFIT);
   PrintFormat("Symbol Currency Base: %s\nSymbol Currency Profit: %s", symbol_currency

//--- 在循环中遍历服务器上所有可用的交易品种
   int total=SymbolsTotal(false), pos=-1;
   for(int i=0; i<total; i++)
     {
      //--- 获取下一个交易品种的名称
      string name=SymbolName(i, false);

      //--- 在交易品种名称中查找包含基础货币名称的子字符串，和
      //--- 如果找到子字符串，则在日志中显示交易品种名称、其在货币列表中的索引以及搜索到的货币名称
      pos = StringFind(name, symbol_currency_base);
      if(pos >= 0)
         PrintFormat("The '%s' symbol at index %u in the list contains the '%s' currer
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 在交易品种名称中查找包含报价货币名称的子字符串，和
   //--- 如果找到子字符串，则在日志中显示交易品种名称、其在货币列表中的索引以及搜索到的货币名称
   pos = StringFind(name, symbol_currency_profit);
   if(pos >= 0)
      PrintFormat("The '%s' symbol at index %u in the list contains the '%s' currer
  }

/*
结果
 StringFind (EURUSD,D1)   Symbol Currency Base: EUR
 StringFind (EURUSD,D1)   Symbol Currency Profit: USD
 The 'EURUSD' symbol at index 0 in the list contains the 'EUR' currency. Substring p
 The 'EURUSD' symbol at index 0 in the list contains the 'USD' currency. Substring p
 The 'GBPUSD' symbol at index 1 in the list contains the 'USD' currency. Substring p
 The 'USDCHF' symbol at index 2 in the list contains the 'USD' currency. Substring p
 The 'USDJPY' symbol at index 3 in the list contains the 'USD' currency. Substring p
 The 'USDCNH' symbol at index 4 in the list contains the 'USD' currency. Substring p
 The 'USDRUB' symbol at index 5 in the list contains the 'USD' currency. Substring p
 The 'AUDUSD' symbol at index 6 in the list contains the 'USD' currency. Substring p
 The 'NZDUSD' symbol at index 7 in the list contains the 'USD' currency. Substring p
 The 'USDCAD' symbol at index 8 in the list contains the 'USD' currency. Substring p
 The 'USDSEK' symbol at index 9 in the list contains the 'USD' currency. Substring p
 The 'USDHKD' symbol at index 10 in the list contains the 'USD' currency. Substring
 The 'USDSGD' symbol at index 11 in the list contains the 'USD' currency. Substring
 The 'USDNOK' symbol at index 12 in the list contains the 'USD' currency. Substring
 The 'USDDKK' symbol at index 13 in the list contains the 'USD' currency. Substring
 The 'USDTRY' symbol at index 14 in the list contains the 'USD' currency. Substring
 The 'USDZAR' symbol at index 15 in the list contains the 'USD' currency. Substring
 The 'USDCZK' symbol at index 16 in the list contains the 'USD' currency. Substring
 The 'USDHUF' symbol at index 17 in the list contains the 'USD' currency. Substring
 The 'USDPLN' symbol at index 18 in the list contains the 'USD' currency. Substring
 The 'USDRUR' symbol at index 19 in the list contains the 'USD' currency. Substring
 The 'EURAUD' symbol at index 27 in the list contains the 'EUR' currency. Substring
 The 'EURCAD' symbol at index 28 in the list contains the 'EUR' currency. Substring
 The 'EURCHF' symbol at index 29 in the list contains the 'EUR' currency. Substring
 The 'EURCZK' symbol at index 30 in the list contains the 'EUR' currency. Substring
 The 'EURDKK' symbol at index 31 in the list contains the 'EUR' currency. Substring
 The 'EURGBP' symbol at index 32 in the list contains the 'EUR' currency. Substring
 The 'EURHKD' symbol at index 33 in the list contains the 'EUR' currency. Substring
 The 'EURHUF' symbol at index 34 in the list contains the 'EUR' currency. Substring
 The 'EURJPY' symbol at index 35 in the list contains the 'EUR' currency. Substring
 The 'EURNOK' symbol at index 36 in the list contains the 'EUR' currency. Substring
 The 'EURNZD' symbol at index 37 in the list contains the 'EUR' currency. Substring
 The 'EURPLN' symbol at index 38 in the list contains the 'EUR' currency. Substring
 The 'EURRUR' symbol at index 39 in the list contains the 'EUR' currency. Substring
 The 'EURRUB' symbol at index 40 in the list contains the 'EUR' currency. Substring
 The 'EURSEK' symbol at index 41 in the list contains the 'EUR' currency. Substring
 The 'EURTRY' symbol at index 42 in the list contains the 'EUR' currency. Substring
```

© 2000-2025, MetaQuotes Ltd.

```
   The 'EURZAR' symbol at index 43 in the list contains the 'EUR' currency. Substring
   The 'XAUUSD' symbol at index 47 in the list contains the 'USD' currency. Substring
   The 'XAUEUR' symbol at index 48 in the list contains the 'EUR' currency. Substring
   The 'XAGUSD' symbol at index 50 in the list contains the 'USD' currency. Substring
   The 'XAGEUR' symbol at index 51 in the list contains the 'EUR' currency. Substring
   The 'USDCRE' symbol at index 53 in the list contains the 'USD' currency. Substring
   The 'XPDUSD' symbol at index 65 in the list contains the 'USD' currency. Substring
   The 'XPTUSD' symbol at index 66 in the list contains the 'USD' currency. Substring
   The 'USDGEL' symbol at index 67 in the list contains the 'USD' currency. Substring
   The 'USDMXN' symbol at index 68 in the list contains the 'USD' currency. Substring
   The 'EURMXN' symbol at index 69 in the list contains the 'EUR' currency. Substring
   The 'USDCOP' symbol at index 75 in the list contains the 'USD' currency. Substring
   The 'USDARS' symbol at index 76 in the list contains the 'USD' currency. Substring
   The 'USDCLP' symbol at index 77 in the list contains the 'USD' currency. Substring
   The 'EURSGD' symbol at index 89 in the list contains the 'EUR' currency. Substring
   The 'USDILS' symbol at index 95 in the list contains the 'USD' currency. Substring
   The 'USDTHB' symbol at index 122 in the list contains the 'USD' currency. Substring
   The 'USDRMB' symbol at index 123 in the list contains the 'USD' currency. Substring
   The 'EURILS' symbol at index 126 in the list contains the 'EUR' currency. Substring
   The 'EURCNH' symbol at index 137 in the list contains the 'EUR' currency. Substring
   The 'USDBRL' symbol at index 139 in the list contains the 'USD' currency. Substring
  */
  }
```

另见

StringSubstr、StringGetCharacter、StringLen、StringLen

© 2000-2025, MetaQuotes Ltd.

# StringGetCharacter

从字符串指定位置返回交易品种的值。

```
ushort  StringGetCharacter(
   string  string_value,     // 字符串
   int     pos               // 字符串中交易品种位置
   );
```

参量

*string_value*

[in] 字符串

*pos*

[in] 字符串中交易品种的位置，能够从0到 StringLen(text) -1.

返回值

交易品种代码或错误为0，获得 错误代码 函数调用 GetLastError().

示例：

```
void OnStart()
  {
//--- 删除图表上的所有评论
   Comment("");
//--- 声明一个字符串，从中获取交易品种代码值并记住字符串的长度
   string message = "The script demonstrates the operation of the StringGetCharacter()
   int    length  = StringLen(message);
//--- 声明一个字符串变量，我们将向其中添加从演示字符串中获得的交易品种
   string text    = "";
//--- 按照演示字符串长度进行循环
   for(int i=0; i<length; i++)
     {
      //--- 等待1/10 秒
      Sleep(100);
      //--- 从位于演示字符串中循环索引处的字符串获取交易品种
      ushort char_code=StringGetCharacter(message, i);
      //--- 将交易品种添加到显示的字符串中，并将结果字符串显示为图表评论
      text+=ShortToString(char_code);
      Comment(text);
     }
//--- 等待两秒钟，然后从图表中删除评论
   Sleep(2000);
   Comment("");

   /*
   Result: the demo string appears on the screen character by character
   The script demonstrates the operation of the StringGetCharacter() function
   */
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另 见

StringSetCharacter、StringBufferLen、StringLen、StringFill、StringInit、StringToCharArray、StringToShortArray

© 2000-2025, MetaQuotes Ltd.

# StringInit

通过指定交易品种初始化一组字符串并提供指定字符串大小。

```
bool  StringInit(
   string&    string_var,        // 要初始化的字符串
   int        new_len=0,         // 初始化后所需要的字符串长度
   ushort     character=0        // 填充字符串的交易品种
   );
```

参量

*string_var*

[in][out] 字符串应该初始化和非初始化。

*new_len=0*

[in] 在初始化结束后的字符串长度，如果长度是0，就没有进行字符串初始化，例如，清除字符串的缓冲器，将缓冲器的地址调到0位置。

*character=0*

[in] 将交易品种填充字符串。

返回值

如果成功返回，显示true,否则是false，获得 错误代码 函数调用 GetLastError() 。

注释

如果字符是0，长度new_len>0，固定位置长度的字符串缓冲器会分散，并以0填满，字符串长度等于0，因为全部缓冲器通过字符串终端填满。

示例：

```
void OnStart()
  {
//---
   string str;
   StringInit(str,200,0);
   Print("str = ",str,": StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/*  Result
str = : StringBufferLen(str) = 200   StringLen(str) = 0
*/
```

另见

StringBufferLen, StringLen

© 2000-2025, MetaQuotes Ltd.

# StringLen

在字符串中返回交易品种数字。

```
int  StringLen(
   string  string_value      // 字符串
   );
```

参量

*string_value*

[in] 字符串计算长度。

返回值

在字符串中交易品种数字不能以0结尾。

示例：

```
void OnStart()
  {
//--- 定义测试字符串
   string text="123456789012345";
//--- 获取字符串中的交易品种数
   int str_len=StringLen(text);
//--- 在日志中显示字符串和其中的交易品种数
   PrintFormat("The StringLen() function returned the value of %d chars in string: '%s

  /*
  结果
   The StringLen() function returned the value of 15 chars in string: '12345678901234
  */
  }
```

另见

StringBufferLen、StringTrimLeft、StringTrimRight、StringToCharArray、StringToShortArray

© 2000-2025, MetaQuotes Ltd.

# StringSetLength

设置一个字符串的指定长度（以字符为单位）。

```
bool  StringSetLength(
   string&     string_var,       // 字符串
   uint        new_length        // 新字符串长度
   );
```

参数

*string_var*

[in][out] 应该用于设置字符新长度的字符串。

*new_capacity*

[in] 以字符为单位的所需字符串长度。如果new_length小于当前大小，则丢弃多余字符。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

StringSetLength()函数不会更改为字符串分配的缓冲区大小。

示例：

```
void OnStart()
  {
//--- 定义字符串
   string text="123456789012345";

//--- 在日志中显示字符串及其长度
   PrintFormat("Before StringSetLength() the string '%s' has a size of %d characters",

//--- 将字符串大小减少到10个字符
   StringSetLength(text, 10);

//--- 将因StringSetLength()操作而变化的字符串及其新长度显示在日志中
   PrintFormat("After StringSetLength() the string is now '%s', and has a size of %d c

  /*
  结果
   Before StringSetLength() the string '123456789012345' has a size of 15 characters
   After StringSetLength() the string is now '1234567890', and has a size of 10 charac
  */
  }
```

另见

StringLen、StringBufferLen、StringReserve StringInit、StringSetCharacter

© 2000-2025, MetaQuotes Ltd.

# StringReplace

通过交易品种集合序列替代字符串发现的所有子字符串。

```
int  StringReplace(
   string&          str,              // 子字符串将被替代的字符串
   const string     find,             // 搜索的子字符串
   const string     replacement       // 子字符串将被插入到发现位置
   );
```

参数

*str*

[in][out] 您将替代子字符串所在的字符串。

*find*

[in] 所需替换的子字符串。

*replacement*

[in] 将被插入的字符串替代所发现的字符串。

返回值

如果成功，函数返回替代的数量，否则为 -1。要获得错误代码，请调用GetLastError()函数。

注意

如果函数成功运行但是没有替换（没有找到替换的子字符串），则返回0。

错误可能从不正确的*str*或*find*参数产生（空字符串或未初始化字符串，请见StringInit()）。此外，如果没有足够的内存完成替换，也会发生错误。

例如：

```
  string text="The quick brown fox jumped over the lazy dog.";
  int replaced=StringReplace(text,"quick","slow");
  replaced+=StringReplace(text,"brown","black");
  replaced+=StringReplace(text,"fox","bear");
  Print("Replaced: ", replaced,". Result=",text);

//  结果
//  替换:3. Result=The slow black bear jumped over the lazy dog.
//
```

另见

StringSetCharacter(), StringSubstr()

© 2000-2025, MetaQuotes Ltd.

# StringReserve

保留内存中字符串的指定大小缓冲区。

```
bool  StringReserve(
   string&     string_var,        // 字符串
   uint        new_capacity       // 存储字符串的缓冲区大小
   );
```

参数

*string_var*

[in][out] 用于更改缓冲区大小的字符串。

*new_capacity*

[in] 字符串所需的缓冲区大小。如果new_capacity大小小于字符串长度，那么当前缓冲区大小不会更改。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

通常，字符串大小不等于用于存储字符串的缓冲区大小。当创建字符串时，通常为相应的缓冲区分配一个空白区。StringReserve()函数可以管理缓冲区大小以及为以后操作指定最佳大小。

不同于 StringInit()，StringReserve()函数不更改字符串的内容，也不填充字符。

例如：

```
void OnStart()
  {
   string s;
//--- 不使用StringReserve检查操作速度
   ulong t0=GetMicrosecondCount();
   for(int i=0; i< 1024; i++)
      s+=" "+(string)i;
   ulong msc_no_reserve=GetMicrosecondCount()-t0;
   s=NULL;
//--- 现在，让我们使用StringReserve执行相同操作
   StringReserve(s,1024 * 3);
   t0=GetMicrosecondCount();
   for(int i=0; i< 1024; i++)
      s+=" "+(string)i;
   ulong msc_reserve=GetMicrosecondCount()-t0;
//--- 检查时间
   Print("Test with StringReserve passed for "+(string)msc_reserve+" msc");
   Print("Test without StringReserve passed for "+(string)msc_no_reserve+" msc");
/* Result:
     Test with StringReserve passed for 50 msc
```

© 2000-2025, MetaQuotes Ltd.

```
      Test without StringReserve passed for 121 msc
*/
  }
```

另 见

StringBufferLen, StringSetLength, StringInit, StringSetCharacter

© 2000-2025, MetaQuotes Ltd.

# StringSetCharacter

在指定位置上返回字符串上复制的改变字节。

```
bool  StringSetCharacter(
   string&   string_var,        // 字符串
   int       pos,               // 位置
   ushort    character          // 字符
   );
```

参量

*string_var*

[in][out] String.

*pos*

[in] 在字符串上的位置特性，从0开始到 StringLen （文本）

*character*

[in] 交易品种代码Unicode。

返回值

成功返回true,否则是false,为了获得 错误代码，可以调用 GetLastError() 函数。

注释

如果销售点少于 字符串长度 并且交易品种代码值是0，切断字符串 （但是 缓冲区大小 ，为未改变的字符串类型分散），字符串长度与销售点相同。

如果*pos*与字符串长度相同，指定交易品种会添加到字符串末尾，长度会一个接一个扩大。

示例：

```
void OnStart()
  {
   string str="0123456789";
   Print("before: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- 在中心添加零值
   StringSetCharacter(str,6,0);
   Print("  after: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- 在末尾添加交易品种
   int size=StringLen(str);
   StringSetCharacter(str,size,'+');
   Print("addition: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/* Result
   before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 10
    after: str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
   addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
```

© 2000-2025, MetaQuotes Ltd.

```
*/
```

另 见

StringBufferLen, StringLen, StringFill, StringInit, CharToString, ShortToString, CharArrayToString, ShortArrayToString

© 2000-2025, MetaQuotes Ltd.

# StringSplit

在指定字符串通过指定分隔符得到子字符串，返回所得子字符串的数量

```
int  StringSplit(
   const string   string_value,        // 字符串搜索
   const ushort   separator,           // 被搜索子字符串使用的分隔符
   string         & result[]           // 通过引用传递数组得到寻找的子字符串
   );
```

参数

*string_value*

[in] 您需要获得子字符串的字符串。字符串不能改变。

*pos*

[in] 分隔符的字符代码。要获得代码，您可以使用 StringGetCharacter() 函数。

*result[]*

[out] 获得的子字符串所在的字符串数组。

返回值

result[]数组中的子字符串数量。如果在过去的字符串没有找到分隔符，那么将只有源字符串位于数组中。

如果string_value 为空或NULL，函数将返回零。如果发生错误，则函数返回-1.若要获得错误代码，请调用GetLastError()函数。

例如：

```
string to_split="_life_is_good_"; // 字符串分成子字符串
   string sep="_";                // 分隔符为字符
   ushort u_sep;                  // 分隔符字符代码
   string result[];               // 获得字符串数组
   //--- 获得分隔符代码
   u_sep=StringGetCharacter(sep,0);
   //--- 字符串分为子字符串
   int k=StringSplit(to_split,u_sep,result);
   //--- 显示注释
   PrintFormat("Strings obtained: %d. Used separator '%s' with the code %d",k,sep,u_se
   //--- 现在输出所有获得的字符串
   if(k>0)
     {
      for(int i=0;i<k;i++)
        {
         PrintFormat("result[%d]=\"%s\"",i,result[i]);
        }
     }
```

另见

StringReplace(), StringSubstr(), StringConcatenate()

© 2000-2025, MetaQuotes Ltd.

# StringSubstr

子串的摘要从指定位置的文本字符串开始。

```
string  StringSubstr(
   string  string_value,      // 字符串
   int     start_pos,         // 初始位置
   int     length=-1          // 提取的字符串的长度
   );
```

参量

*string_value*

[in] 字符串提取子串。

*start_pos*

[in] 子串的初始位置，从0到 StringLen(文本)-1。

*length=-1*

[in] 提取子串的长度，如果参量值是-1或者没有建立参量，子串会从指定位置提取直到字符串结束。

返回值

复制提取的子串，如果可能的话，否则返回空字符串

示例：

```
void OnStart()
  {
//--- 获取当前交易品种的名称
   string name = Symbol();

//--- 获取交易品种基础货币和报价货币
   string base   = StringSubstr(name, 0, 3);
   string quoted = StringSubstr(name, 3, 3);

//--- 在日志中显示获得的交易品种货币
   PrintFormat("Symbol: %s. Currency base: %s, currency quoted: %s", name, base, quote

  /*
  结果
   Symbol: EURUSD. Currency base: EUR, currency quoted: USD
  */
  }
```

另见

StringSplit、StringFind、StringGetCharacter

© 2000-2025, MetaQuotes Ltd.

# StringToLower

所选字符串的所有交易品种通过存储单元转变成小写字母。

```
bool  StringToLower(
   string&  string_var      // 要处理的字符串
   );
```

参量

*string_var*

[in][out] 字符串。

返回值

如果成功，返回值是true,否则是false，获得 错误代码 调用 GetLastError()。

示例：

```
void OnStart()
  {
//--- 用大写字母定义源字符串
   string text=" - THIS STRING, WRITTEN IN UPPERCASE, MUST BE WRITTEN IN LOWERCASE";
//--- 在日志中显示源字符串
   Print("Source line:\n", text);
//--- 将所有字符串的字符转换为小写字母并在日志中显示结果
   if(StringToLower(text))
      Print("The original string after using the StringToLower() function:\n", text);

  /*
  结果
   Source line:
    - THIS STRING, WRITTEN IN UPPERCASE, MUST BE WRITTEN IN LOWERCASE
   The original string after using the StringToLower() function:
    - this string, written in uppercase, must be written in lowercase
  */
  }
```

另见

StringToUpper、StringTrimLeft、StringTrimRight

© 2000-2025, MetaQuotes Ltd.

# StringToUpper

所有选择字符串中的交易品种通过存储单元转换成大写字母。

```
bool  StringToUpper(
   string&  string_var      // 要处理的字符串
   );
```

参量

*string_var*

[in][out] 字符串

返回值

成功返回真值，否则是失败值，获得 错误代码 调用 GetLastError()。

示例：

```
void OnStart()
  {
//--- 用小写字母定义源字符串
   string text=" - this string, written in lowercase, must be written in uppercase";
//--- 在日志中显示源字符串
   Print("Source line:\n", text);
//--- 将所有字符串的字符转换为大写字母并在日志中显示结果
   if(StringToUpper(text))
      Print("The original string after using the StringToUpper() function:\n", text);

  /*
  结果
   Source line:
    - this string, written in lowercase, must be written in uppercase
   The original string after using the StringToUpper() function:
    - THIS STRING, WRITTEN IN LOWERCASE, MUST BE WRITTEN IN UPPERCASE
  */
  }
```

另见

StringToLower、StringTrimLeft、StringTrimRight

© 2000-2025, MetaQuotes Ltd.

# StringTrimLeft

函数为字符换行，在第一位的实意交易品种后，是字符串左边的字符和标号，字符串适当修改。

```
int  StringTrimLeft(
   string&  string_var      // 要剪切的字符串
   );
```

参量

*string_var*

[in][out] 字符串从左边剪切。

返回值

返回剪切交易品种数量。

示例：

```
void OnStart()
  {
//--- 定义源字符串 左侧六个空格）
   string text="      All spaces on the left will be removed from this string";
//--- 在日志中显示源字符串
   PrintFormat("Source line:\n'%s'", text);
//--- 删除左侧的所有空格，并在日志中显示删除的字符数和结果字符串
   int num=StringTrimLeft(text);
   PrintFormat("The StringTrimLeft() function removed %d chars from the left side. Now

  /*
  结果
   Source line:
   '      All spaces on the left will be removed from this string'
   The StringTrimLeft() function removed 6 chars from the left side. Now the line look
   'All spaces on the left will be removed from this string'
  */
  }
```

另见

StringTrimRight、StringToLower、StringToUpper

© 2000-2025, MetaQuotes Ltd.

# StringTrimRight

函数为字符换行，在最后面的实意交易品种后，是字符串右边的字符和标号，字符串适当修改。

```
int  StringTrimRight(
   string&  string_var      // 要剪切的字符串
   );
```

参量

*string_var*

[in][out] 字符串从右边剪切。

返回值

返回剪切交易品种数量。

示例：

```
void OnStart()
  {
//--- 定义源字符串 右侧六个空格）
   string text="All spaces on the right will be removed from this string      ";
//--- 在日志中显示源字符串
   PrintFormat("Source line:\n'%s'", text);
//--- 删除右侧的所有空格，并在日志中显示删除的字符数和结果字符串
   int num=StringTrimRight(text);
   PrintFormat("The StringTrimRight() function removed %d chars from the right side. N

  /*
  结果
   Source line:
   'All spaces on the right will be removed from this string      '
   The StringTrimRight() function removed 6 chars from the right side. Now the line lo
   'All spaces on the right will be removed from this string'
  */
  }
```

另见

StringTrimLeft、StringToLower、StringToUpper

© 2000-2025, MetaQuotes Ltd.

# 日期和时间

函数组与 日期时间 类型一起工作（整型代表已使用的秒数从1970.1.1.零时开始）。

要安排高分辨率的计数器和计时器，可使用GetTickCount()函数，它产生的值以毫秒为单位。

| 函数 | 功能 |
| --- | --- |
| TimeCurrent | 日期时间格式返回服务器最后一个可知时间（最后一个报价收据时间） |
| TimeTradeServer | 返回交易服务器当前计算时间 |
| TimeLocal | 以日期时间格式返回本地计算机时间 |
| TimeGMT | 通过客户端运行的计算机本地时间以日期时间格式返回GMT日节省时间。 |
| TimeDaylightSavings | 返回日节省时间切换符号 |
| TimeGMTOffset | 以秒计算返回当前GMT和计算机本地的不同，也包括夏令时切换 |
| TimeToStruct | 日期时间值转换到MqlDateTime结构类型变量 |
| StructToTime | MqlDateTime结构类型变量转化到日期时间值 |

© 2000-2025, MetaQuotes Ltd.

# TimeCurrent

返回最后访问的服务器时间，最后在"市场报价"窗口中选择交易品种接收的引用时间。在 OnTick() 处理器中，函数返回接收的处理器时间。在其他情况下（例如，调用 handlers OnInit(), OnDeinit(), OnTimer()等等），对于在"市场报价"窗口中的交易品种常量有 最后一个报价收据时间，时间在窗口标题上显示。时间值在交易服务器上形成并不能依靠时间设置电脑。有两种变量函数。

无参量调用

```
datetime  TimeCurrent();
```

调用MqlDateTime类型参量

```
datetime  TimeCurrent(
   MqlDateTime&  dt_struct      // 结构类型变量
   );
```

参量

*dt_struct*

[out] MqlDateTime 结构类型变量

返回值

日期时间 类型值

注释

如果 MqlDateTime 结构类型变量以参量传递，就会因此填满。

要安排高分辨率的计数器和计时器，可使用 GetTickCount() 函数，它产生的值以毫秒为单位。

在策略测试器进行测试期间，TimeCurrent() 根据历史数据进行模拟。

示例：

```
void OnStart()
  {
//--- 声明MqlDateTime变量以填充日期/时间数据，并从市场报价窗口获取最后报价的时间
   MqlDateTime tm={};
   datetime    time1=TimeCurrent();     // 第一种调用形式:市场报价窗口中某一交易品种的最后报价
   datetime    time2=TimeCurrent(tm);   // 第二种调用形式:填充MqlDateTime结构的市场报价窗口中

//--- 显示接收日期/时间的结果，并在日志中使用相应的数据填充结构
   PrintFormat("Tick time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u\n-
               (string)time1, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.day
   /*
   Result:
   Tick time: 2024.04.18 15:40:06
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 15
```

© 2000-2025, MetaQuotes Ltd.

```
   - Min: 40
   - Sec: 06
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeTradeServer

返回交易服务器计算出的当前时间，不像TimeCurrent()，时间值的计算在客户端中执行并依据电脑里的时间设置。有两种常量函数。

无参量调用

```
datetime  TimeTradeServer();
```

调用MqlDateTime类型参量

```
datetime  TimeTradeServer(
   MqlDateTime&  dt_struct      // 结构类型变量
   );
```

参量

*dt_struct*

[out] MqlDateTime 结构类型变量

返回值

日期时间 类型值

注释

如果MqlDateTime如果结构类型变量以参量传递，会因此填满。

要安排高分辨率的计数器和计时器，可使用GetTickCount()函数，它产生的值以毫秒为单位。

在策略测试器进行测试期间，TimeTradeServer()总是相当于模拟服务器时间的TimeCurrent()。

示例：

```
void OnStart()
  {
//--- 声明MqlDateTime变量以填充日期/时间数据，并获取最后报价的时间和估算的交易服务器当前时间
   MqlDateTime tm={};
   datetime    time_current=TimeCurrent();                  // 第一种调用形式:市场报价窗口
   datetime    time_server =TimeTradeServer(tm);            // 第二种调用形式:通过填充Mql
   int         difference  =int(time_current-time_server);  // 当前时间和交易服务器时间的

//--- 使用日志中填充的MqlDateTime结构的数据显示最后报价的时间和估算的交易服务器当前时间
   PrintFormat("Time Current: %s\nTime Trade Server: %s\n- Year: %u\n- Month: %02u\n-
               "- Hour: %02u\n- Min: %02u\n- Sec: %02u\n- Day of Year: %03u\n- Day of
               (string)time_current, (string)time_server, tm.year, tm.mon, tm.day, tm.
               EnumToString((ENUM_DAY_OF_WEEK)tm.day_of_week), difference);
   /*
   Result:
   Time Current: 2024.04.18 16:10:14
   Time Trade Server: 2024.04.18 16:10:15
   - Year: 2024
   - Month: 04
```

© 2000-2025, MetaQuotes Ltd.

```
   - Day: 18
   - Hour: 16
   - Min: 10
   - Sec: 15
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   Difference between Time Current and Time Trade Server: -1
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeLocal

返回客户端运行时电脑上的当地时间，有两种变量函数。

无参量调用

```
datetime  TimeLocal();
```

调用MqlDateTime类型参量

```
datetime  TimeLocal(
   MqlDateTime&  dt_struct      // 结构类型变量
   );
```

参量

*dt_struct*

[out] MqlDateTime 结构类型变量

返回值

日期时间 类型值

注释

如果MqlDateTime结构类型变量以参量传递，就会填满。

要安排高分辨率的计数器和计时器，可使用GetTickCount()函数，它产生的值以毫秒为单位。

在策略测试器进行测试期间，TimeLocal()总是相当于模拟服务器时间的 TimeCurrent() 。

示例：

```
void OnStart()
  {
//--- 声明MqlDateTime变量以填充PC本地时间数据
   MqlDateTime tm={};
   datetime    time1=TimeLocal();   // 第一种调用形式:PC本地时间
   datetime    time2=TimeLocal(tm); // 第二种调用形式:填充MqlDateTime结构的PC本地时间

//--- 显示接收PC本地时间的结果，并在日志中使用相应的数据填充结构
   PrintFormat("Local time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u\r
               (string)time1, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.day
   /*
   Result:
   Local time: 2024.04.18 19:44:09
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 19
   - Min: 44
   - Sec: 09
   - Day of Year: 108
```

© 2000-2025, MetaQuotes Ltd.

```
   - Day of Week: 4 (THURSDAY)
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeGMT

返回GMT，该计算考虑到当客户端运行时电脑上当地时间的DST转换，有两种变量函数。

无参量调用

```
datetime  TimeGMT();
```

调用MqlDateTime类型参量

```
datetime  TimeGMT(
   MqlDateTime&  dt_struct      // 结构类型变量
   );
```

参量

*dt_struct*

[out] MqlDateTime 结构类型变量。

返回值

日期时间 类型值

注释

如果MqlDateTime结构类型变量以参量传递，就会填满。

要安排高分辨率的计数器和计时器，可使用GetTickCount()函数，它产生的值以毫秒为单位。

在策略测试器进行测试期间，TimeGMT()总是相当于模拟服务器时间的 TimeTradeServer()。

示例：

```
void OnStart()
  {
//--- 声明MqlDateTime变量以填充日期/时间数据，并获取PC本地时间和GMT
   MqlDateTime tm={};
   datetime    time1=TimeLocal();            // 第一种调用形式:PC本地时间
   datetime    time2=TimeGMT(tm);            // 第二种调用形式:通过填充MqlDateTime结构从PC
   int         shift=int(time1-time2)/3600;  // 本地时间相对于GMT的偏移量

//--- 使用日志中填充的MqlDateTime结构的数据显示本地时间和GMT
   PrintFormat("Time Local: %s\nTime GMT: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n
               "- Hour: %02u\n- Min: %02u\n- Sec: %02u\n- Day of Year: %03u\n- Day of
               (string)time1, (string)time2, tm.year, tm.mon, tm.day, tm.hour, tm.min,
               EnumToString((ENUM_DAY_OF_WEEK)tm.day_of_week), shift);
   /*
   Result:
   Time Local: 2024.04.18 19:37:23
   Time GMT: 2024.04.18 12:37:23
   - Year: 2024
   - Month: 04
   - Day: 18
```

© 2000-2025, MetaQuotes Ltd.

```
   - Hour: 12
   - Min: 37
   - Sec: 23
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   Local time offset relative to GMT: +7
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeDaylightSaving

返回以秒为单位的正确时间，转变成夏季时间，取决于电脑上的时间设置。

```
int  TimeDaylightSavings();
```

返回值

如果转换成冬季（标准的）时间，返回0。

示例：

```
void OnStart()
  {
//--- 以秒数为单位获取夏令时调整
   int sec_dl=TimeDaylightSavings();

//--- 创建描述接收值的文本
   string text=(sec_dl==0 ? "Standard \"winter\" time is used" :
                StringFormat("Daylight saving time has been switched over. The correct

//--- 在日志中以秒数为单位显示夏令时调整的描述信息
   Print(text);
   /*
   Result for "winter" time:
   Standard "winter" time is used

   result for "summer" time:
   Daylight saving time has been switched over. The correction is -3600 seconds
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeGMTOffset

返回在GMT时间和当前电脑之间的不同，考虑到转换成冬季或者夏季时间，取决于电脑上设置的时间。

```
int  TimeGMTOffset();
```

返回值

整型值表示以秒数计算的GMT时间和电脑本地时间TimeLocal之间的当前差异。

```
TimeGMTOffset() =  TimeGMT() - TimeLocal()
```

示例：

```
void OnStart()
  {
//--- 获取本地时间，GMT和GMT与本地计算机时间之间的差值（秒）
   datetime time_local=TimeLocal();
   datetime time_gmt  =TimeGMT();
   int      offset    =TimeGMTOffset();

//--- 在日志中显示获得的时间和偏移值
   PrintFormat("Local Time: %s, GMT Time: %s, Seconds Offset: %+d", (string)time_local
   /*
   Result:
   Local Time: 2024.04.18 19:35:52, GMT Time: 2024.04.18 12:35:52, Seconds Offset: -25
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TimeToStruct

转变时间类型值（数字从1970.01.01以秒开始）到结构变量 MqlDateTime 中。

```
bool  TimeToStruct(
   datetime      dt,            // 日期和时间
   MqlDateTime&  dt_struct      // 采用值的结构
   );
```

参量

*dt*

[in] 转变数据值。

*dt_struct*

[out] 结构类型MqlDateTime变量

返回值

如果成功返回true，否则false。若要获得有关错误的信息，请调用GetLastError() 函数。

示例：

```
void OnStart()
  {
//--- 获取服务器的最后已知时间，声明日期/时间结构并填充结构字段
   datetime    time=TimeCurrent();
   MqlDateTime tm  ={};
   if(!TimeToStruct(time,tm))
      Print("TimeToStruct() failed. Error ", GetLastError());

//--- 在日志中显示获得的服务器时间和使用TimeToStruct()填充MqlDateTime结构的结果
   PrintFormat("Server time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u\
               (string)time, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.day_
   /*
   Result:
   Server time: 2024.04.18 15:47:27
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 15
   - Min: 47
   - Sec: 27
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# StructToTime

转变结构变量 MqlDateTime 到 日期时间 类型值，然后返回结果值。

```
datetime  StructToTime(
   MqlDateTime&  dt_struct      // 日期和时间结构
   );
```

参量

*dt_struct*

[in] 结构类型变量MqlDateTime。

返回值

日期时间型的值包括从01.01.1970起的每秒钟数据。

示例：

```
#property script_show_inputs

input int   InpYear  =  0;    // 年
input int   InpMonth =  0;    // 月
input int   InpDay   =  0;    // 日
input int   InpHour  =  0;    // 小时
input int   InpMin   =  0;    // 分钟
input int   InpSec   =  0;    // 秒

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 调整输入值并将其写入变量
   int year =  (InpYear<1970 ?1970 : InpYear); // 如果输入的年份早于1970，则使用1970
   int month=  (InpMonth<1 ?  1  :  InpMonth > 12  ?  12 :  InpMonth);
   int day  =  (InpDay  <1 ?  1  :  InpDay   > 31  ?  31 :  InpDay);
   int hour =  (InpHour <0 ?  0  :  InpHour  > 23  ?  23 :  InpHour);
   int min  =  (InpMin  <0 ?  0  :  InpMin   > 59  ?  59 :  InpMin);
   int sec  =  (InpSec  <0 ?  0  :  InpSec   > 59  ?  59 :  InpSec);

//--- 在日志中显示输入的值
   PrintFormat("Entered date and time: %04u.%02u.%02u %02u:%02u:%02u", InpYear, InpMor

//--- 在日志中显示调整后的输入值
   PrintFormat("Corrected date and time: %04u.%02u.%02u %02u:%02u:%02u", year, month,

//--- 将输入值写入相应的结构字段
   MqlDateTime time_struct={};
   time_struct.year= year;
   time_struct.mon = month;
```

© 2000-2025, MetaQuotes Ltd.

```
   time_struct.day = day;
   time_struct.hour= hour;
   time_struct.min = min;
   time_struct.sec = sec;

//--- 将结构中的日期和时间转换为datetime类型的变量，和
   datetime time = StructToTime(time_struct);

//--- 显示从MqlDateTime结构类型变量到datetime类型值的转换结果
   Print("Converted date and time: ",TimeToString(time,TIME_DATE|TIME_MINUTES|TIME_SEC
   /*
   Results if zero default values are entered:
   Entered date and time: 0000.00.00 00:00:00
   Corrected date and time: 1970.01.01 00:00:00
   Converted date and time: 1970.01.01 00:00:00

   results if the wrong day of the current year's February is entered:
   Entered date and time: 2024.02.31 00:00:00
   Corrected date and time: 2024.02.31 00:00:00
   Converted date and time: 2024.03.02 00:00:00
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# 账户信息

返回当前账户参量的函数。

| 函数 | 功能 |
| --- | --- |
| AccountInfoDouble | 返回相应账户属性的双精度值 |
| AccountInfoInteger | 返回相应账户属性的整数类型值（布尔，整型或者长整型） |
| AccountInfoString | 返回相应账户属性的字符串类型值 |

© 2000-2025, MetaQuotes Ltd.

# AccountInfoDouble

返回对应账户财产值。

```
double  AccountInfoDouble(
   ENUM_ACCOUNT_INFO_DOUBLE  property_id      // 属性标识符
   );
```

参量

*property_id*

[in] 属性指标，值可以是ENUM_ACCOUNT_INFO_DOUBLE其中一个

返回值

双精度 类型值

示例：

```
void OnStart()
  {
//--- 显示AccountInfoDouble()函数中所有有效信息
   printf("ACCOUNT_BALANCE =  %G",AccountInfoDouble(ACCOUNT_BALANCE));
   printf("ACCOUNT_CREDIT =  %G",AccountInfoDouble(ACCOUNT_CREDIT));
   printf("ACCOUNT_PROFIT =  %G",AccountInfoDouble(ACCOUNT_PROFIT));
   printf("ACCOUNT_EQUITY =  %G",AccountInfoDouble(ACCOUNT_EQUITY));
   printf("ACCOUNT_MARGIN =  %G",AccountInfoDouble(ACCOUNT_MARGIN));
   printf("ACCOUNT_MARGIN_FREE =  %G",AccountInfoDouble(ACCOUNT_MARGIN_FREE));
   printf("ACCOUNT_MARGIN_LEVEL =  %G",AccountInfoDouble(ACCOUNT_MARGIN_LEVEL));
   printf("ACCOUNT_MARGIN_SO_CALL = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL));
   printf("ACCOUNT_MARGIN_SO_SO = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_SO));
  }
```

另见

SymbolInfoDouble， SymbolInfoString， SymbolInfoInteger， PrintFormat

© 2000-2025, MetaQuotes Ltd.

# AccountInfoInteger

返回账户属性值。

```
long  AccountInfoInteger(
   ENUM_ACCOUNT_INFO_INTEGER  property_id      // 属性标识符
   );
```

参量

*property_id*

[in] 属性标识符，值可以是 ENUM_ACCOUNT_INFO_INTEGER值中一个。

返回值

长整型 类型值

整数

属性可以是 布尔, 整型 或者 长整型 类型中的一个。

示例：

```
void OnStart()
  {
//--- 显示AccountInfoInteger()函数中所有有效信息
   printf("ACCOUNT_LOGIN =  %d",AccountInfoInteger(ACCOUNT_LOGIN));
   printf("ACCOUNT_LEVERAGE =  %d",AccountInfoInteger(ACCOUNT_LEVERAGE));
   bool thisAccountTradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_ALLOWED);
   bool EATradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_EXPERT);
   ENUM_ACCOUNT_TRADE_MODE tradeMode=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOU
   ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoIntege

//--- 通知完成交易操作的可能性
   if(thisAccountTradeAllowed)
      Print("Trade for this account is permitted");
   else
      Print("Trade for this account is prohibited!");

//--- 找出是否可能通过EA交易进行这个账户的交易
   if(EATradeAllowed)
      Print("Trade by Expert Advisors is permitted for this account");
   else
      Print("Trade by Expert Advisors is prohibited for this account!");

//--- 找出账户类型
   switch(tradeMode)
     {
      case(ACCOUNT_TRADE_MODE_DEMO):
         Print("This is a demo account");
         break;
      case(ACCOUNT_TRADE_MODE_CONTEST):
```

© 2000-2025, MetaQuotes Ltd.

```
         Print("This is a competition account");
         break;
      default:Print("This is a real account!");
     }

//--- 找出止损离场水平设置模式
   switch(stopOutMode)
     {
      case(ACCOUNT_STOPOUT_MODE_PERCENT):
         Print("The StopOut level is specified percentage");
         break;
      default:Print("The StopOut level is specified in monetary terms");
     }
  }
```

另 见

账户信息

© 2000-2025, MetaQuotes Ltd.

# AccountInfoString

返回对应账户属性值。

```
string  AccountInfoString(
   ENUM_ACCOUNT_INFO_STRING  property_id      // 属性标识符
   );
```

参数

*property_id*

[in] 属性标识符，值可以是 ENUM_ACCOUNT_INFO_STRING值中一个。

返回值

字符串 类型值

示例：

```
void OnStart()
  {
//--- 显示AccountInfoString()函数中所有有效信息
   Print("The name of the broker = ",AccountInfoString(ACCOUNT_COMPANY));
   Print("Deposit currency = ",AccountInfoString(ACCOUNT_CURRENCY));
   Print("Client name = ",AccountInfoString(ACCOUNT_NAME));
   Print("The name of the trade server = ",AccountInfoString(ACCOUNT_SERVER));
  }
```

另见

账户信息

© 2000-2025, MetaQuotes Ltd.

# 启动检测

函数返回当前客户端现行状态的参量

| 函数 | 功能 |
| --- | --- |
| GetLastError | 返回上一个错误 |
| IsStopped | 如果mq5程序被令停止操作，返回true |
| UninitializeReason | 返回无法初始化原因代码 |
| TerminalInfoInteger | 返回mq5程序环境相关属性的整数值 |
| TerminalInfoDouble | 返回mq5程序环境相关属性的双精度值 |
| TerminalInfoString | 返回mq5程序环境相关属性的字符串值 |
| MQLInfoInteger | 返回运行mq5程序相关属性的整数值 |
| MQLInfoString | 返回运行mq5程序相关属性的字符串值 |
| Symbol | 返回当前图表的交易品种的名称 |
| Period | 返回当前图表时间表 |
| Digits | 返回决定当前图表交易品种价格值的精确性的小数位数 |
| Point | 返回报价货币当前交易品种的点大小 |

© 2000-2025, MetaQuotes Ltd.

# GetLastError

返回 _LastError系统变量目录。

```
int  GetLastError();
```

返回值

在MQL5程序执行时，返回最后发生的错误值。

注释

在函数调用后，不重设_LastError目录。重设变量，需要调用 ResetLastError()。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlRates rates[1]={};   // 在这里显示当前柱形图数据

//--- 故意使用不合适的参数调用函数
   int res=CopyRates(NULL, PERIOD_CURRENT, 0, 2, rates);
   if(res!=2)
      PrintFormat("CopyRates() returned %d. LastError %d", res, GetLastError());

//--- 将当前柱形图数据复制到MqlRates结构之前，请重置最后一个错误代码
   ResetLastError();
//--- 如果函数不能正常工作，错误代码将不等于0
   CopyRates(NULL, PERIOD_CURRENT, 0, 1, rates);
   Print("CopyRates() error ", GetLastError());

//--- 打印获得的值的数组
   ArrayPrint(rates);
  }
```

另见

交易服务器返回代码

© 2000-2025, MetaQuotes Ltd.

# IsStopped

检测MQL5程序的强制关机。

```
bool  IsStopped();
```

返回值

如果系统变量包括非0值，返回true，非零值录入_StopFlag，如果MQL5程序命令完成操作，在此情况下，必须立即终止程序，否则系统在3秒后从外部强制完成。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 无限循环并停止检查
   while(!IsStopped())
     {
      //--- 在图表上显示本地电脑时间
      Comment("Time Local: ", TimeToString(TimeLocal(), TIME_DATE|TIME_MINUTES|TIME_SE
      Sleep(16);
     }
   Print("The StopFlag is set. The program will be stopped.");

//--- 清除
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# UninitializeReason

返回代码 无法初始化原因 。

```
int  UninitializeReason();
```

返回值

在 _UninitReason调用之前返回OnDeinit() 值，该值依靠导致无法初始化的原因。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 将去初始化的原因代码放入'reason'变量中
   int reason=UninitializeReason();
//--- 创建带有去初始化原因的消息字符串，并将该消息发送到日志
   string message=StringFormat("%s: Uninitialize reason code: %d (%s)",__FUNCTION__, r
   Print(message);
//--- 成功
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 从'reason'形式变量中创建带有去初始化原因的消息字符串，并将该消息发送到日志
   string message=StringFormat("%s: Uninitialize reason code: %d (%s)",__FUNCTION__, r
   Print(message);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
  }
//+------------------------------------------------------------------+
//| 返回去初始化原因代码的描述                                       |
//+------------------------------------------------------------------+
string UninitializeReasonDescription(const int reason)
  {
   switch(reason)
     {
      //--- EA已停止调用ExpertRemove()函数的工作
      case REASON_PROGRAM :
        return("Expert Advisor terminated its operation by calling the ExpertRemove()
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 程序从图表移除
      case REASON_REMOVE :
        return("Program has been deleted from the chart");
      //--- 程序重新编译
      case REASON_RECOMPILE :
        return("Program has been recompiled");
      //--- 交易品种或图表周期被改变
      case REASON_CHARTCHANGE :
        return("Symbol or chart period has been changed");
      //--- 图表已关闭
      case REASON_CHARTCLOSE :
        return("Chart has been closed");
      //--- 由用户更改的输入参数
      case REASON_PARAMETERS :
        return("Input parameters have been changed by a user");
      //--- 由于账户设置的变化，另一个账户已被激活或已被重新连接到交易服务器
      case REASON_ACCOUNT :
        return("Another account has been activated or reconnection to the trade server
      //--- 应用另一个图表模板
      case REASON_TEMPLATE :
        return("A new template has been applied");
      //--- OnInit()处理程序返回一个非零值
      case REASON_INITFAILED :
        return("This value means that OnInit() handler has returned a nonzero value");
      //--- 程序端已关闭
      case REASON_CLOSE :
        return("Terminal has been closed");
     }

//--- 去初始化原因未知
   return("Unknown reason");
  }
```

© 2000-2025, MetaQuotes Ltd.

# TerminalInfoInteger

在MQL5程序环境中该函数返回一定值。

```
int  TerminalInfoInteger(
   int  property_id      // 属性标识符
   );
```

参量

*property_id*

[in]属性标识符，或许是计算式值 ENUM_TERMINAL_INFO_INTEGER中一个。

返回值

整型值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取正在运行的程序端的构建号及其“64位程序端”属性
   int  build = TerminalInfoInteger(TERMINAL_BUILD);
   bool x64   = TerminalInfoInteger(TERMINAL_X64);

//--- 在日志中记录获得的程序端数据
   PrintFormat("MetaTrader 5 %s build %d", (x64 ? "x64" : "x32"), build);
   /*
   result:
   MetaTrader 5 x64 build 4330
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TerminalInfoDouble

返回mql5程序环境相应属性的值。

```
double  TerminalInfoDouble(
   int  property_id      // 属性的标识符
   );
```

参数

*property_id*

[in]属性标识符。可以是枚举ENUM_TERMINAL_INFO_DOUBLE值的一个。

返回值

双精度类型的值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取正在运行的程序端的构建号及其“64位程序端”属性
   int  build = TerminalInfoInteger(TERMINAL_BUILD);
   bool x64   = TerminalInfoInteger(TERMINAL_X64);

//--- 在日志中记录获得的程序端数据
   PrintFormat("MetaTrader 5 %s build %d", (x64 ? "x64" : "x32"), build);
   /*
   result:
   MetaTrader 5 x64 build 4330
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# TerminalInfoString

在MQL5程序环境中该函数返回一定值，该属性一定是字符串型。

```
string  TerminalInfoString(
   int  property_id      // 属性标识符
   );
```

参量

*property_id*

[in]属性标识符，或许是计算式值 ENUM_TERMINAL_INFO_STRING中一个。

返回值

字符串类型值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取操作系统和程序端数据
   string os_ver  = TerminalInfoString(TERMINAL_OS_VERSION);       // 用户的操作系统
   string name    = TerminalInfoString(TERMINAL_NAME);             // 程序端名称
   string path    = TerminalInfoString(TERMINAL_PATH);             // 启动程序端的文件夹
   string data    = TerminalInfoString(TERMINAL_DATA_PATH);        // 存储程序端数据的文件
   string common  = TerminalInfoString(TERMINAL_COMMONDATA_PATH); // 计算机上安装的所有客

//--- 将获得的数据发送到日志
   PrintFormat("OS: %s\nTerminal: %s\n- Path: %s\n- Data path: %s\n- Common Data path:
   /*
   OS: Windows 10 build 19045
   Terminal: MetaTrader 5
   - Path: E:\MetaQuotes\MetaTrader 5
   - Data path: E:\MetaQuotes\MetaTrader 5
   - Common Data path: C:\Users\admin\AppData\Roaming\MetaQuotes\Terminal\Common
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# MQLInfoInteger

返回运行的MQL程序的类似属性值。

```
int  MQLInfoInteger(
   int  property_id      // 属性标识符
   );
```

参量

*property_id*

[in]属性标识符，可以是 ENUM_MQL_INFO_INTEGER 值中的一个。

返回值

整型值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取MQL程序的可用内存和已消耗内存的值
   int limit = MQLInfoInteger(MQL_MEMORY_LIMIT);    // MQL5程序的最大动态内存量（以MB为单位）
   int used  = MQLInfoInteger(MQL_MEMORY_USED);     // MQL5程序使用的内存大小（以MB为单位）

//--- 将接收到的值发送到日志
   PrintFormat("Maximum possible amount of dynamic memory for MQL5 program: %d Mb\n"+
               "Memory used by MQL5 program: %d Mb", limit, used);
   /*
   result
   Maximum possible amount of dynamic memory for MQL5 program: 8388608 Mb
   Memory used by MQL5 program: 2 Mb
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# MQLInfoString

返回运行中的MQL5程序的类似属性值。

```
string  MQLInfoString(
   int  property_id      // 属性标识符
   );
```

参量

*property_id*

[in]属性标识符，可以是 ENUM_MQL_INFO_STRING 值中的一个。

返回值

字符串类型值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取启动的程序数据
   string name = MQLInfoString(MQL_PROGRAM_NAME);  // 已启动的MQL5程序的名称
   string path = MQLInfoString(MQL_PROGRAM_PATH);  // 运行程序路径

//--- 将获得的数据发送到日志
   PrintFormat("Name of the running MQL program: '%s'\nPath of the running MQL progra
   /*
   result:
   Name of the running MQL program: 'MQLInfoString'
   Path of the running MQL program: E:\MetaQuotes\MetaTrader 5\MQL5\Scripts\MQLInfoSt
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# 交易品种

返回当前图表的交易品种名称。

```
string  Symbol();
```

返回值

_Symbol 系统变量的值，存储当前图表交易品种的名称。

注意

与EA交易、指标和脚本不同，服务并不绑定特定的图表。因此，Symbol()为一项服务返回一个空字符串("")。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取当前图表交易品种名称
   string name = Symbol();

//--- 将获得的数据发送到日志
   PrintFormat("Current chart symbol name: '%s'", name);
   /*
   result
   Current chart symbol name: 'EURUSD'
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# Period

返回当前图表的时间表。

```
ENUM_TIMEFRAMES  Period();
```

返回值

_Period 变量内容包括当前图表时间表的值，该值可以是 ENUM_TIMEFRAMES 计算式中的一个。

注意

与EA交易、指标和脚本不同，服务并不绑定特定的图表。因此，Period()为服务返回0。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取当前图表的时间表值及其描述
   ENUM_TIMEFRAMES period    = Period();
   string          timeframe = StringSubstr(EnumToString(period), 7);

//--- 将获得的数据发送到日志
   PrintFormat("Current chart timeframe: %s\nTimeframe value: %s (%d)",
               timeframe, EnumToString(period), period);
   /*
   result:
   Current chart timeframe: H4
   Timeframe value: PERIOD_H4 (16388)
   */
  }
```

另见

PeriodSeconds，图表时间表，日期和时间，对象可见性

© 2000-2025, MetaQuotes Ltd.

# Digits

返回当前图表交易品种价格精确度的小数位数数量。

```
int  Digits();
```

返回值

_Digits 变量值存储在小数位数数量中，觉得当前图表交易品种价格的精确度。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取当前图表交易品种的小数位数
   int digits = Digits();

//--- 将获得的数据发送到日志
   Print("Number of decimal digits for the current chart symbol: ", digits);
   /*
   result:
   Number of decimal digits for the current chart symbol: 5
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# Point

在引用货币中返回当前交易品种大小点。

```
double  Point();
```

返回值

_Point 变量值，在引用货币中存储当前交易品种的磅值。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取以报价货币表示的当前交易品种点大小
   double point = Point();

//--- 将获得的数据发送到日志
   Print("Point size of the current symbol in the quote currency: ", DoubleToString(po
   /*
   result:
   Point size of the current symbol in the quote currency: 0.00001
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# 事件处理

MQL5语言提供对特定预定义事件的处理。处理这些事件的函数应该在MQL5程序中定义：函数名、返回类型、一组参数（如果有），并且它们的类型应该严格地符合事件处理函数的描述。

客户端事件处理程序使用返回和参数类型来识别处理事件的函数。如果某个函数有一些参数或者返回类型与下面的描述不符，那么这样的函数就不可以被用于处理事件。

| 函数 | 描述 |
| --- | --- |
| OnStart | 当Start事件发生时调用这个函数，来执行脚本中设定的操作 |
| OnInit | 当Init事件发生时在指标和EA中调用这个函数，来初始化启动的MQL5程序 |
| OnDeinit | 当Deinit事件发生时在指标和EA中调用这个函数，来去初始化启动的MQL5程序 |
| OnTick | 当NewTick事件发生时在EA中调用这个函数，来处理一个新报价 |
| OnCalculate | 当Calculate事件发生时在指标中调用这个函数，来处理价格数据的变化 |
| OnTimer | 当Timer周期性事件在固定时间间隔内由程序端生成期间，在指标和EA中调用这个函数 |
| OnTrade | 当Trade事件在交易服务器上交易操作结束时生成期间，在EA中调用这个函数 |
| OnTradeTransaction | 当TradeTransaction事件发生时在EA中调用这个函数，来处理一个交易请求执行结果 |
| OnBookEvent | 当BookEvent事件发生时在EA中调用这个函数，来处理市场深度中的变化 |
| OnChartEvent | 当ChartEvent事件发生时在指标和EA中调用这个函数，来处理由用户或者MQL5程序所做的图表更改 |
| OnTester | 当Tester事件发生时在EA中调用这个函数，以便在对历史数据测试EA之后执行必要的操作 |
| OnTesterInit | 当TesterInit事件发生时在EA中调用这个函数，以便在策略测试优化之前执行必要的操作 |
| OnTesterDeinit | 在策略测试EA优化之后，当TesterDeinit事件发生时在EA中调用这个函数。 |
| OnTesterPass | 当TesterPas事件发生时在EA中调用这个函数，以便在策略测试中处理EA优化期间到达的新数据框架 |

客户端将传入事件发送到相应的打开图表中。此外，事件也可以通过图表（图表事件）或mql5程序（自定义事件）来生成。通过设置CHART_EVENT_OBJECT_CREATE和CHART_EVENT_OBJECT_DELETE图表属性可以启用/禁用生成图形对象创建/删除事件。每个mql5应用程序和图表都有其自己的事件队列，所有新到达的事件也都包括其中。

© 2000-2025, MetaQuotes Ltd.

一个程序只从其所运行的图表中获取事件。所有事件都按照它们的接收顺序来处理。如果这个队列已经包括NewTick事件，或者这个事件正在处理阶段，那么新的NewTick事件就不能添加到mql5应用程序队列中。同样，如果ChartEvent已在mql5程序序列或者这种事件正被处理，那么这种类型的新事件都不可放在队列之中。Timer事件处理也以相同的方式进行——如果Timer事件已在队列中或者正被处理中，那么队列中则不设置新的timer事件

事件队列具有一定限制但充足的规模，所以对于一个正确的开发程序来说，不可能出现队列溢出的情况。当队列溢出时，新事件会被舍弃，而不会将其设置为队列。

强烈建议不要使用无限循环来处理事件。可能的例外就是处理单独的Start事件的脚本。

程序库不处理任何事件。

© 2000-2025, MetaQuotes Ltd.

# OnStart

当Start事件发生时在脚本和服务中调用这个函数。这个函数旨在一次性执行程序中实施的操作。有两种函数类型。

返回结果的版本

```
int  OnStart(void);
```

返回值

int类型的值显示在"日志"选项卡中。

脚本执行完成之后，在程序端日志中创建了"脚本script_name已移除（结果代码N）"条目。在这里，N是OnStart()函数的返回值。

服务执行完成之后，在程序端日志中创建了"服务service_name已停止（结果代码N）"条目。在这里，N是OnStart()函数的返回值。

建议使用返回执行结果的OnStart()调用，因为它不仅允许脚本或服务执行，还可以返回错误代码或其他有用的数据来分析程序执行结果。

注意

OnStart()是处理脚本和服务事件的唯一函数。不会有其他事件被发送至这些程序。并且，Start事件不被传递到EA和自定义指标。

没有结果返回的版本只为与旧代码兼容而保留。不建议使用

```
void  OnStart(void);
```

注意

OnStart()是处理脚本和服务事件的唯一函数。不会有其他事件被发送至这些程序。并且，Start事件不被传递到EA和自定义指标。

脚本样例：

```
//--- 用于处理颜色的宏
#define XRGB(r,g,b)     (0xFF000000|(uchar(r)<<16)|(uchar(g)<<8)|uchar(b))
#define GETRGB(clr)     ((clr)&0xFFFFFF)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置一个下行的蜡烛图颜色
   Comment("Set a downward candle color");
   ChartSetInteger(0,CHART_COLOR_CANDLE_BEAR,GetRandomColor());
   ChartRedraw(); // 无需等候新报价，立即更新图表
   Sleep(1000);   // 暂停1秒，查看所有的变化
//--- 设置一个上行的蜡烛图颜色
   Comment("Set an upward candle color");
   ChartSetInteger(0,CHART_COLOR_CANDLE_BULL,GetRandomColor());
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartRedraw();
   Sleep(1000);
//--- 设置背景颜色
   Comment("Set the background color");
   ChartSetInteger(0,CHART_COLOR_BACKGROUND,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置买价线的颜色
   Comment("Set color of Ask line");
   ChartSetInteger(0,CHART_COLOR_ASK,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置卖价线的颜色
   Comment("Set color of Bid line");
   ChartSetInteger(0,CHART_COLOR_BID,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置下行柱形图和下行蜡烛图框架的颜色
   Comment("Set color of a downward bar and a downward candle frame");
   ChartSetInteger(0,CHART_COLOR_CHART_DOWN,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置图表线和Doji蜡烛图的颜色
   Comment("Set color of a chart line and Doji candlesticks");
   ChartSetInteger(0,CHART_COLOR_CHART_LINE,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置上行柱形图和上行蜡烛图框架的颜色
   Comment("Set color of an upward bar and an upward candle frame");
   ChartSetInteger(0,CHART_COLOR_CHART_UP,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置坐标轴、比例和OHLC线的颜色
   Comment("Set color of axes, scale and OHLC line");
   ChartSetInteger(0,CHART_COLOR_FOREGROUND,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置一个网格颜色
   Comment("Set a grid color");
   ChartSetInteger(0,CHART_COLOR_GRID,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置最后价格线
   Comment("Set Last price color");
   ChartSetInteger(0,CHART_COLOR_LAST,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置止损和获利水平的颜色
   Comment("Set color of Stop Loss and Take Profit order levels");
```

© 2000-2025, MetaQuotes Ltd.

```
   ChartSetInteger(0,CHART_COLOR_STOP_LEVEL,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- 设置交易量的颜色和市场进入水平
   Comment("Set color of volumes and market entry levels");
   ChartSetInteger(0,CHART_COLOR_VOLUME,GetRandomColor());
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 返回一个随机生成的颜色                                           |
//+------------------------------------------------------------------+
color GetRandomColor()
  {
   color clr=(color)GETRGB(XRGB(rand()%255,rand()%255,rand()%255));
   return clr;
  }
```

另见

事件处理函数、程序运行、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnInit

当Init事件发生时在指标和EA中调用这个函数。它被用于初始化运行中的MQL5程序。有两种函数类型。

返回结果的版本

```
int  OnInit(void);
```

返回值

int类型值，0意味着成功初始化。

当返回INIT_FAILED时，EA会从图表中强制卸载。

返回INIT_FAILED时，指标不会从图表中卸载。图表上保留的指标无法运行 — 事件处理程序不会在指标中调用。

建议使用返回执行结果的OnInit()调用，因为它不仅可以程序初始化，还可以在早期程序终止的情况下返回一个错误代码。

没有结果返回的版本只为与旧代码兼容而保留。不建议使用

```
void  OnInit(void);
```

注意

加载EA或指标之后会立即生成Init事件。不为脚本生成该事件。OnInit()函数被用来初始化MQL5程序。如果OnInit()有一个int类型的返回值，那么非零的返回代码意味着初始化失败，且生成REASON_INITFAILED去初始化原因代码的Deinit事件。

void类型的OnInit()函数始终表示初始化成功，且不建议使用。

为了优化EA 输入，建议使用来自ENUM_INIT_RETCODE枚举值作为返回代码。这些值旨在建立优化过程管理，包括选择最合适的测试代理。在启动测试之前，在EA初始化过程中，可以使用TerminalInfoInteger()函数请求关于代理配置和资源（核心数量、空闲内存量等）的数据。根据所获得的数据，您既可以允许使用测试代理，也可以禁止它优化EA。

| ID | 描述 |
| --- | --- |
| INIT_SUCCEEDED | 初始化成功，EA测试可以继续。<br>这个代码意同零值——测试中的EA初始化成功。 |
| INIT_FAILED | 初始化失败。由于不可避免的错误，因此继续测试没有意义。例如，不能创建EA操作所需的指标。<br>这个值返回意同返回非零的值——测试中的EA初始化失败。 |
| INIT_PARAMETERS_INCORRECT | 旨在表示程序员一组不正确的输入参数。在通用的优化表格中，该返回代码的结果字符串以红色突出显示。<br>不执行这组EA输入的测试。代理已准备好接收新任务。<br>当收到该值时，策略测试不将此任务传递到另一个代理重复执行。 |
| INIT_AGENT_NOT_SUITABLE | 在初始化过程中没有程序执行错误。然而，出于某些原因，代理不适合进行测试。例如，没有乳沟的RAM，没有OpenCL support等。 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| | 返回该代码之后，直至这个优化结束，代理才会再接收任务。 |

使用OnInit()在测试中返回INIT_FAILED/INIT_PARAMETERS_INCORRECT有一些再优化EA时需要考虑的特性：

- OnInit()返回INIT_PARAMETERS_INCORRECT的参数集被认为不适合进行测试，并且在遗传优化期间不用于获取下一个群集。在寻找最佳EA参数时，太多"废弃"参数集可能导致错误的结果。搜索EA假设优化准则函数平滑，在整个输入参数上没有间隙。
- 如果OnInit()返回INIT_FAILED，这意味着测试无法启动，EA从代理内存中卸载。再次加载EA，以使用一组新参数执行下一个传递。与调用TesterStop()相比，启动下一次优化传递需要更多的时间。

EA的OnInit()样例函数

```
//--- 输入参数
input int      ma_period=20; //移动平均线周期

//--- 在EA中使用的指标句柄
int indicator_handle;
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 检查ma_period有效性
   if(ma_period<=0)
     {
      PrintFormat("Invalid ma_period input value: %d",ma_period);
      return (INIT_PARAMETERS_INCORRECT);
     }
//--- 优化过程中
   if(MQLInfoInteger(MQL_OPTIMIZATION))
     {
      //--- 为代理检查可用的RAM
      int available_memory_mb=TerminalInfoInteger(TERMINAL_MEMORY_TOTAL);
      if(available_memory_mb<2000)
        {
         PrintFormat("Insufficient memory for the test agent: %d MB",
                     available_memory_mb);
         return (INIT_AGENT_NOT_SUITABLE);
        }
     }
//--- 检查指标
   indicator_handle=iCustom(_Symbol,_Period,"My_Indicator",ma_period);
   if(indicator_handle==INVALID_HANDLE)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("Failed to generate My_Indicator handle. Error code %d",
                  GetLastError());
      return (INIT_FAILED);
     }
//--- EA初始化成功
   return(INIT_SUCCEEDED);
  }
```

另见

OnDeinit、事件处理函数、程序运行、客户端事件、变量的初始化、创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# OnDeinit

当Deinit事件发生时在指标和EA中调用这个函数。它被用于去初始化运行中的MQL5程序。

```
void  OnDeinit(
   const int  reason          //去初始化原因代码
   );
```

参数

*reason*

[in] 去初始化原因代码。

返回值

无返回值

注意

在以下情况下，为EA和指标生成Deinit事件：

- 由于mql5程序附加的交易品种或图表周期发生变化而重新初始化之前；
- 由于输入变化而重新初始化之前；
- 卸载mql5程序之前。

*reason*参数可能有以下值：

| 常量 | 值 | 描述 |
|---|---|---|
| REASON_PROGRAM | 0 | EA已停止调用ExpertRemove()函数的工作 |
| REASON_REMOVE | 1 | 程序从图表移除 |
| REASON_RECOMPILE | 2 | 程序重新编译 |
| REASON_CHARTCHANGE | 3 | 交易品种或图表周期被改变 |
| REASON_CHARTCLOSE | 4 | 图表已关闭 |
| REASON_PARAMETERS | 5 | 由用户更改的输入参数 |
| REASON_ACCOUNT | 6 | 由于账户设置的变化，另一个账户已被激活或已被重新连接到交易服务器 |
| REASON_TEMPLATE | 7 | 应用另一个图表模板 |
| REASON_INITFAILED | 8 | OnInit()处理程序返回一个非零值 |
| REASON_CLOSE | 9 | 程序端已关闭 |

EA去初始化原因代码可通过UninitializeReason()函数接收或者从预定义_UninitReason变量接收。

EA的OnInit()和OnDeinit()示例函数

```
input int fake_parameter=3;       // 无用的参数
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int OnInit()
  {
//--- 获取程序被编译的构建号
   Print(__FUNCTION__," Build #",__MQLBUILD__);
//--- 在OnInit()中还可以获得重置原因代码
   Print(__FUNCTION__," Deinitialization reason code can be received during the EA res
//--- 获得去初始化原因代码的第一个方式
   Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
//--- 获得去初始化原因代码的第二个方式
   Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReaso
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 获得去初始化原因代码的第一个方式
   Print(__FUNCTION__," Deinitialization reason code = ",reason);
//--- 获得去初始化原因代码的第二个方式
   Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
//--- 获得去初始化原因代码的第三个方式
   Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReaso
  }
//+------------------------------------------------------------------+
//| 返回去初始化原因代码的文本描述                                   |
//+------------------------------------------------------------------+
string getUninitReasonText(int reasonCode)
  {
   string text="";
//---
   switch(reasonCode)
     {
      case REASON_ACCOUNT:
         text="Account was changed";break;
      case REASON_CHARTCHANGE:
         text="Symbol or timeframe was changed";break;
      case REASON_CHARTCLOSE:
         text="Chart was closed";break;
      case REASON_PARAMETERS:
         text="Input-parameter was changed";break;
      case REASON_RECOMPILE:
         text="Program "+__FILE__+" was recompiled";break;
      case REASON_REMOVE:
         text="Program "+__FILE__+" was removed from chart";break;
      case REASON_TEMPLATE:
         text="New template was applied to chart";break;
      default:text="Another reason";
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//---
   return text;
  }
```

另见

OnInit、事件处理函数、程序运行、客户端事件、去初始化原因代码、可视范围和变量的生命周期、创建和删除对象

© 2000-2025, MetaQuotes Ltd.

# OnTick

当NewTick事件发生时在EA中调用这个函数，来处理一个新报价。

```
void  OnTick(void);
```

返回值

无返回值

注意

NewTick事件只在收到一个新报价时为EA生成，以获得EA所附加的图表的交易品种。在自定义指标或脚本中定义OnTick()函数没有意义，因为没有为其生成NewTick事件。

Tick事件只为EA生成，但这并不意味着EA必须包含OnTick()函数的功能，因为除了NewTick以外，还为EA生成Timer、BookEvent和ChartEvent事件。

所有事件都按照它们的接收顺序依次处理。如果这个队列已经包括NewTick事件，或者这个事件正在处理阶段，那么新的NewTick事件就不能添加到mql5应用程序队列中。

无论是否启用自动交易（自动交易按键），都会生成NewTick事件。禁用自动交易意味着只禁止从EA发送交易请求。EA操作不会停止。

通过按下自动交易按键来禁用自动交易并不会中断OnTick()函数的当前执行。

在OnTick()函数中，显示整个交易逻辑特性的EA示例

```
//+------------------------------------------------------------------+
//|                                                  TradeByATR.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA trading in the \"explosive\" candle direction"
#property description "\"Explosive\" candle has the body size exceeding k*ATR"
#property description "The \"revers\" parameter reverses the signal direction"

input double lots=0.1;         // 交易量手数
input double kATR=3;           // ATR中的信号蜡烛长度
input int    ATRperiod=20;     // ATR指标周期
input int    holdbars=8;       // 保持持仓的柱形图数量
input int    slippage=10;      // 可允许滑移
input bool   revers=false;     //反向信号?
input ulong  EXPERT_MAGIC=0;   // EA幻数
//--- 用于存储ATR指标句柄
int atr_handle;
//--- 在这里，我们将存储ATR最后值和蜡烛体
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 初始化全局变量
   last_atr=0;
   last_body=0;
//--- 设置正确的交易量
   double min_lot=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   trade_lot=lots>min_lot? lots:min_lot;
//--- 创建ATR指标句柄
   atr_handle=iATR(_Symbol,_Period,ATRperiod);
   if(atr_handle==INVALID_HANDLE)
     {
      PrintFormat("%s: failed to create iATR, error code %d",__FUNCTION__,GetLastError
      return(INIT_FAILED);
     }
//--- EA成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 通知EA操作结束代码
   Print(__FILE__,": Deinitialization reason code = ",reason);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 交易信号
   static int signal=0; // +1表示买入信号，-1 表示卖出信号
//--- 检查和关闭 'holdbars'柱形图之前开仓的旧持仓
   ClosePositionsByBars(holdbars,slippage,EXPERT_MAGIC);
//--- 检查新柱形图
   if(isNewBar())
     {
      //--- 检查信号是否存在
      signal=CheckSignal();
     }
//--- 如果持有单边持仓，跳过信号-等候直至关闭
   if(signal!=0 && PositionsTotal()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
     {
      signal=0;
      return; // 退出NewTick事件处理程序，并在新柱形图出现之前不要进入市场
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 对于锁仓账户，每个持仓都是单独持仓和平仓
   if(signal!=0)
     {
      //--- 买入信号
      if(signal>0)
        {
         PrintFormat("%s: Buy signal! Revers=%s",__FUNCTION__,string(revers));
         if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
      //--- 卖出信号
      if(signal<0)
        {
         PrintFormat("%s: Sell signal! Revers=%s",__FUNCTION__,string(revers));
         if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
     }
//--- OnTick函数结束
  }
//+------------------------------------------------------------------+
//| 检查新的交易信号                                                 |
//+------------------------------------------------------------------+
int CheckSignal()
  {
//--- 0 意味着没有信号
   int res=0;
//--- 获得倒数第二个完整柱形图的ATR值（柱形图索引是2）
   double atr_value[1];
   if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
     {
      last_atr=atr_value[0];
      //--- 获得最后关闭的柱形图的MqlRates类型数组数据
      MqlRates bar[1];
      if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
        {
         //--- 计算最后完整柱形图的柱体大小
         last_body=bar[0].close-bar[0].open;
         //--- 如果最后柱形图的柱体（索引为1）超过的之前ATR的值（柱形图索引为2），那么会收到一个交易信
         if(MathAbs(last_body)>kATR*last_atr)
            res=last_body>0?1:-1; // 向上蜡烛图的正值
        }
      else
         PrintFormat("%s: Failed to receive the last bar! Error",__FUNCTION__,GetLastE
     }
   else
      PrintFormat("%s: Failed to receive ATR indicator value! Error",__FUNCTION__,GetL
//--- 如果启用了反向交易模式
```

© 2000-2025, MetaQuotes Ltd.

```
   res=revers?-res:res;  // 在必要时反转信号 返回-1而不是1，反之亦然）
//--- 返回一个交易信号值
   return (res);
  }
//+------------------------------------------------------------------+
//|  当新柱形图出现时返回'true'                                      |
//+------------------------------------------------------------------+
bool isNewBar(const bool print_log=true)
  {
   static datetime bartime=0; //存储当前柱形图的开盘时间
//--- 获得零柱的开盘时间
   datetime currbar_time=iTime(_Symbol,_Period,0);
//--- 如果开盘时间更改，则新柱形图出现
   if(bartime!=currbar_time)
     {
      bartime=currbar_time;
      lastbar_timeopen=bartime;
      //--- 在日志中显示新柱形图开盘时间的数据
      if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION)||MQLInfoInteger(MQL_TESTER)))
        {
         //--- 显示新柱形图开盘时间的信息
         PrintFormat("%s: new bar on %s %s opened at %s",__FUNCTION__,_Symbol,
                     StringSubstr(EnumToString(_Period),7),
                     TimeToString(TimeCurrent(),TIME_SECONDS));
         //--- 获取关于最后报价的数据
         MqlTick last_tick;
         if(!SymbolInfoTick(Symbol(),last_tick))
            Print("SymbolInfoTick() failed, error = ",GetLastError());
         //--- 显示最后报价的时间，精确至毫秒
         PrintFormat("Last tick was at %s.%03d",
                     TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%1000
        }
      //--- 我们有一个新柱形图
      return (true);
     }
//--- 没有新柱形图
   return (false);
  }
//+------------------------------------------------------------------+
//| 以指定交易量和市场价买入                                         |
//+------------------------------------------------------------------+
bool Buy(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- 以市场价买入
   return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| 以指定交易量和市场价卖出                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool Sell(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- 以市场价卖出
   return (MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| 在柱形图中根据持有时间平仓                                       |
//+------------------------------------------------------------------+
void ClosePositionsByBars(int holdtimebars,ulong deviation=10,ulong  magicnumber=0)
  {
   int total=PositionsTotal(); // 持仓数量
//--- 重复持仓
   for(int i=total-1; i>=0; i--)
     {
      //--- 持仓参数
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
      int bars=iBarShift(_Symbol,PERIOD_CURRENT,position_open)+1;

      //--- 如果持仓的生命周期很长，则幻数和交易品种匹配
      if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
         int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
         double volume=PositionGetDouble(POSITION_VOLUME);
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         string str_type=StringSubstr(EnumToString(type),14);
         StringToLower(str_type); //纠正消息格式的小写文本案例
         PrintFormat("Close position #%I64u %s %s %.2f",
                     position_ticket,position_symbol,str_type,volume);
         //--- 设置一个订单类型并发送交易请求
         if(type==POSITION_TYPE_BUY)
            MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber,position_ticket);
         else
            MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber,position_ticket);
        }
     }
  }
//+------------------------------------------------------------------+
//| 准备和发送交易请求                                               |
//+------------------------------------------------------------------+
bool MarketOrder(ENUM_ORDER_TYPE type,double volume,ulong slip,ulong magicnumber,ulong
  {
//--- 声明和初始化结构
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(type==ORDER_TYPE_BUY)
```

© 2000-2025, MetaQuotes Ltd.

```
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- 请求参数
   request.action   =TRADE_ACTION_DEAL;                    // 交易操作类型
   request.position =pos_ticket;                           // 关闭情况下的持仓单号
   request.symbol   =Symbol();                             // 交易品种
   request.volume   =volume;                               // 交易量
   request.type     =type;                                 // 订单类型
   request.price    =price;                                // 交易价格
   request.deviation=slip;                                 // 可允许的价格偏差
   request.magic    =magicnumber;                          // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
     {
      //--- 显示数据失败
      PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
                  request.symbol,EnumToString(type),volume,request.price,GetLastError
      return (false);
     }
//--- 通知成功操作
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
   return (true);
  }
```

另见

事件处理函数、程序运行、客户端事件、OnTimer、OnBookEvent、OnChartEvent

© 2000-2025, MetaQuotes Ltd.

# OnCalculate

当Calculate事件发生时在指标中调用这个函数，来处理价格数据的变化。有两种函数类型。一个指标内只可以使用其中一种。

基于数据数组的计算

```
int  OnCalculate(
   const int        rates_total,       // price[]数组大小
   const int        prev_calculated,   // 在前一个调用中处理过的柱形图数量
   const int        begin,             //price[]数组中，有意义数据开始的索引编号
   const double&    price[]            // 计算值数组
   );
```

基于当前时间周期时间序列的计算

```
int  OnCalculate(
   const int        rates_total,       // 输入时间序列的大小
   const int        prev_calculated,   // 在前一个调用中处理过的柱形图数量
   const datetime&  time[],            // 时间数组
   const double&    open[],            // 开盘价数组
   const double&    high[],            // 最高价数组
   const double&    low[],             // 最低价数组
   const double&    close[],           // 收盘价数组
   const long&      tick_volume[],     // 报价量数组
   const long&      volume[],          // 真实交易量数组
   const int&       spread[]           // 点差数组
   );
```

参数

*rates_total*

[in] 可用于指标计算的price[]数组或输入序列的大小。在第二个函数类型中，当参数值对应于图表中柱形图数量时，启动这种函数类型。

*prev_calculated*

[in] 包含上一次调用期间OnCalculate()函数的返回值。它的设计旨在跳过自该函数上次启动后没有改变的柱形图。

*begin*

[in] price[]数组中，有意义数据开始的索引编号。它允许您跳过缺失或初始数据，因为没有正确的值。

*price[]*

[in] 计算值数组。其中一个价格时间序列或计算过的指标缓冲区可以作为price[]数组进行传递。为计算而传递的数据类型可以使用_AppliedTo预定义变量来定义。

*time[]*

[in] 柱形图开盘时间值的数组。

*open[]*

[in] 开盘价格值的数组。

© 2000-2025, MetaQuotes Ltd.

*high[]*
[in] 最高价格值的数组。

*low[]*
[in] 最低价格值的数组。

*close[]*
[in] 收盘价格值的数组。

*tick_volume[]*
[in] 报价量值的数组。

*volume[]*
[in] 交易量值的数组。

*spread[]*
[in] 柱形图点差值的数组。

### 返回值

int类型值将在下一次函数调用期间作为*prev_calculated*参数被传递。

### 注意

如果OnCalculate()函数等于零，则没有指标值显示在客户的数据窗口中。

如果价格数据在最后一次调用OnCalculate()函数之后发生了变化（已加载更深层历史记录或已填充历史记录空白），那么程序端本身则将*prev_calculated*输入参数设为零。

若要在*time[]*、*open[]*、*high[]*、*low[]*、*close[]*、*tick_volume[]*、*volume[]*和*spread[]*数组中定义索引方向，请调用ArrayGetAsSeries()函数。为了不依赖于默认值，请为数组调用ArraySetAsSeries()函数来进行处理。

当使用第一种函数类型时，在启动指标时用户选择必要的时间序列或指标作为“参数”选项卡中的price[]数据。为此，请在“应用于”字段的下拉列表中指定必要的元素。

若要获取其他mql5程序的自定义指标值，则使用iCustom()函数。它返回后续操作的指标句柄。也可以指定所需的*price []*数组或另一个指标的句柄。这个参数应被传递到自定义指标输入变量列表的最后。

有必要使用OnCalculate()函数返回值和*prev_calculated*第二个输入参数之间的连接。当调用函数时，*prev_calculated*函数包含了上一次调用期间OnCalculate()函数的返回值。这使得为计算自定义指标而实现资源节约算法成为可能，以避免重复计算自上一次启动该函数以来没有改变的柱形图。

### 样例指标

```
//+------------------------------------------------------------------+
//|                                          OnCalculate_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"
#property description "Sample Momentum indicator calculation"

//---- 指标设置
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  Blue
//---- 输入
input int MomentumPeriod=14; // 计算周期
//---- 指标缓存区
double    MomentumBuffer[];
//--- 存储计算周期的全局变量
int       IntPeriod;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 检查输入参数
   if(MomentumPeriod<0)
     {
      IntPeriod=14;
      Print("Period parameter has an incorrect value. The following value is to be use
     }
   else
      IntPeriod=MomentumPeriod;
//---- 缓冲区
   SetIndexBuffer(0,MomentumBuffer,INDICATOR_DATA);
//---- 显示在“数据窗口”和子窗口的指标名称
   IndicatorSetString(INDICATOR_SHORTNAME,"Momentum"+"("+string(IntPeriod)+")");
//--- 设置柱形图绘制的开始索引
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,IntPeriod-1);
//--- 将0.0设置为未绘制的空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//--- 显示的指标精确性
   IndicatorSetInteger(INDICATOR_DIGITS,2);
  }
//+------------------------------------------------------------------+
//|  动量指标计算                                                    |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,      // price[] array size
                const int prev_calculated, // 之前处理的柱形图的数量
                const int begin,           //重要数据起点
                const double &price[])     // 值数组处理
  {
//--- 初始持仓计算
```

© 2000-2025, MetaQuotes Ltd.

```
   int StartCalcPosition=(IntPeriod-1)+begin;
//---- 如果计算数据不足
   if(rates_total<StartCalcPosition)
      return(0);  // 以零值推出 - 这个指标没有被计算
//--- 正确绘制开始
   if(begin>0)
      PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,StartCalcPosition+(IntPeriod-1));
//--- 开始计算，定义开始位置
   int pos=prev_calculated-1;
   if(pos<StartCalcPosition)
      pos=begin+IntPeriod;
//--- 主计算循环
   for(int i=pos;i<rates_total && !IsStopped();i++)
      MomentumBuffer[i]=price[i]*100/price[i-IntPeriod];
//--- OnCalculate执行完成。返回后续调用的新prev_calculated值
   return(rates_total);
  }
```

另见

ArrayGetAsSeries、ArraySetAsSeries、iCustom、事件处理函数、程序运行、客户端事件、访问时间序列和指标

© 2000-2025, MetaQuotes Ltd.

# OnTimer

当Timer事件在固定时间间隔内由程序端生成期间，在EA中调用这个函数。

```
void  OnTimer(void);
```

返回值

无返回值

注意

Timer事件是由客户端为EA定期生成，使用EventSetTimer()函数来激活计时器。通常，这个函数在OnInit()函数中调用。EA停止工作时，应使用通常在OnDeinit()函数中被调用的EventKillTimer()消除计时器。

每个EA和每个指标都使用其自己的计时器，并从该计时器单独接收事件。mql5程序关闭期间，如果计时器已经创建但没有被EventKillTimer()函数禁用，那么计时器则被强制销毁。

如果您不满足于每秒一次，需要更频繁地接收计时器事件，请使用EventSetMillisecondTimer()来创建高分辨率计时器。

一般来说，当计时器周期减少时，测试时间会增加，因为计时器事件的处理程序会更频繁地被调用。在实时模式下工作时，由于硬件的限制，计时器事件在10-16毫秒内生成不超过1次。

每个程序只能启动一个计时器。每个mql5应用程序和图表都有其自己的事件队列，所有新到达的事件也都包括其中。如果这个队列已经包括Timer事件，或者这个事件正在处理阶段，那么新的Timer事件就不能添加到mql5应用程序队列中。

EA样例，OnTimer()处理程序

```
//+------------------------------------------------------------------+
//|                                                OnTimer_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Example of using the timer for calculating the trading server t
#property description "It is recommended to run the EA at the end of a trading week be
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建一个1秒周期的计时器
   EventSetTimer(1);

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| EA交易去初始化函数                                                |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 完成工作后销毁计时器
   EventKillTimer();

  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer函数                                                        |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- OnTimer()首次调用的时间
   static datetime start_time=TimeCurrent();
//--- 第一次调用OnTimer()期间交易服务器的时间
   static datetime start_tradeserver_time=0;
//--- 计算交易服务器时间
   static datetime calculated_server_time=0;
//--- 本地PC时间
   datetime local_time=TimeLocal();
//--- 目前估计的交易服务器时间
   datetime trade_server_time=TimeTradeServer();
//--- 如果由于某种原因而未知服务器时间，则提前退出
   if(trade_server_time==0)
      return;
//--- 如果尚未设置初始交易服务器的值
   if(start_tradeserver_time==0)
     {
      start_tradeserver_time=trade_server_time;
      //--- 设置交易服务器的计算值
      Print(trade_server_time);
      calculated_server_time=trade_server_time;
     }
   else
     {
      //--- 增加OnTimer()第一次调用的时间
      if(start_tradeserver_time!=0)
         calculated_server_time=calculated_server_time+1;;
     }
//---
```

© 2000-2025, MetaQuotes Ltd.

```
   string com=StringFormat("                  Start time: %s\r\n",TimeToString(start_t
   com=com+StringFormat("                   Local time: %s\r\n",TimeToString(local_time
   com=com+StringFormat("TimeTradeServer time: %s\r\n",TimeToString(trade_server_time,
   com=com+StringFormat(" EstimatedServer time: %s\r\n",TimeToString(calculated_server
//--- 显示图表上所有计数器的值
   Comment(com);
  }
```

另见

EventSetTimer、EventSetMillisecondTimer、EventKillTimer、GetTickCount、GetMicrosecondCount、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnTrade

当Trade事件发生时在EA中调用这个函数。这个函数旨在处理订单、持仓和交易列表的变化。

```
void  OnTrade(void);
```

返回值

无返回值

注意

OnTrade()只为EA交易调用。即使您在指标和脚本中添加了具有相同名称和类型的函数，也不能在这些指标和脚本中使用此函数。

对于任何交易操作（下达挂单、开仓/平仓、下止损单、激活挂单等），订单和交易的历史和/或持仓和当前订单的列表都被适当更改。

当处理一个订单时，交易服务器向程序端发送一条关于传入Trade事件的消息。若要从历史记录中检索订单和交易的相关数据，需要先使用HistorySelect()执行交易历史请求。

如果出现以下情况，则服务器生成交易事件：

- 更改活跃订单，
- 更改持仓，
- 更改交易，
- 更改交易历史。

每一个 Trade事件都可能作为一个或多个交易请求的结果出现。交易请求使用OrderSend()或OrderSendAsync()，被发送到服务器。每个请求都可能导致多个交易事件。您不能依赖“一个请求，一个交易事件”的声明，因为事件的处理可能分为几个阶段执行，并且每个操作都可能更改订单的状态、持仓和交易历史记录。

OnTrade()处理程序在适当调用OnTradeTransaction()之后被调用。通常，在OnTrade ()和OnTradeTransaction ()调用的数量上没有确切的相关性。OnTrade()一次调用对应一次或多次的OnTradeTransaction调用。

EA样例，OnTrade()处理程序

```
//+------------------------------------------------------------------+
//|                                                OnTrade_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

input    int days=7;             //按天计算的交易历史深度
//--- 在全局范围内设置交易历史的界限
datetime     start;              // 缓存中交易历史的开始日期
datetime     end;                // 缓存中交易历史的结束日期
//--- 全局计数器
```

© 2000-2025, MetaQuotes Ltd.

```
int           orders;           // 活跃订单数量
int           positions;        // 持仓数量
int           deals;            // 交易历史缓存中的交易数量
int           history_orders;   // 交易历史缓存中的订单数量
bool          started=false;    // 计数器相关性标识

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   end=TimeCurrent();
   start=end-days*PeriodSeconds(PERIOD_D1);
   PrintFormat("Limits of the history to be loaded: start - %s, end - %s",
               TimeToString(start),TimeToString(end));
   InitCounters();
//---
   return(0);
  }
//+------------------------------------------------------------------+
//|  持仓、订单和交易计数器的初始化                                  |
//+------------------------------------------------------------------+
void InitCounters()
  {
   ResetLastError();
//--- 加载历史
   bool selected=HistorySelect(start,end);
   if(!selected)
     {
      PrintFormat("%s. Failed to load history from %s to %s to cache. Error code: %d",
                  __FUNCTION__,TimeToString(start),TimeToString(end),GetLastError());
      return;
     }
//--- 获取当前值
   orders=OrdersTotal();
   positions=PositionsTotal();
   deals=HistoryDealsTotal();
   history_orders=HistoryOrdersTotal();
   started=true;
   Print("Counters of orders, positions and deals successfully initialized");
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
   if(started) SimpleTradeProcessor();
   else InitCounters();
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 当Trade事件到达时被调用                                          |
//+------------------------------------------------------------------+
void OnTrade()
  {
   if(started) SimpleTradeProcessor();
   else InitCounters();
  }
//+------------------------------------------------------------------+
//| 处理交易和历史变化的示例                                         |
//+------------------------------------------------------------------+
void SimpleTradeProcessor()
  {
   end=TimeCurrent();
   ResetLastError();
//--- 从指定的时间间隔下载交易历史到程序缓存
   bool selected=HistorySelect(start,end);
   if(!selected)
     {
      PrintFormat("%s. Failed to load history from %s to %s to cache. Error code: %d",
                  __FUNCTION__,TimeToString(start),TimeToString(end),GetLastError());
      return;
     }
//--- 获取当前值
   int curr_orders=OrdersTotal();
   int curr_positions=PositionsTotal();
   int curr_deals=HistoryDealsTotal();
   int curr_history_orders=HistoryOrdersTotal();
//--- 检查活跃订单的数量是否发生变化
   if(curr_orders!=orders)
     {
      //--- 活跃订单的数量已经发生变化
      PrintFormat("Number of orders has been changed. Previous value is %d, current va
                  orders,curr_orders);
      //--- 更新值
      orders=curr_orders;
     }
//--- 持仓数量的变化
   if(curr_positions!=positions)
     {
      //--- 持仓的数量已经发生变化
      PrintFormat("Number of positions has been changed. Previous value is %d, current
                  positions,curr_positions);
      //--- 更新值
      positions=curr_positions;
     }
//--- 交易历史缓存中的交易数量的变化
   if(curr_deals!=deals)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 交易历史缓存中的交易数量已经发生变化
       PrintFormat("Number of deals has been changed. Previous value is %d, current val
                   deals,curr_deals);
       //--- 更新值
       deals=curr_deals;
      }
//--- 交易历史缓存中的历史订单数量的变化
   if(curr_history_orders!=history_orders)
      {
       //--- 交易历史缓存中的历史订单数量已经发生变化
       PrintFormat("Number of orders in history has been changed. Previous value is %d,
                   history_orders,curr_history_orders);
      //--- 更新值
      history_orders=curr_history_orders;
      }
//--- 检查是否有必要更改缓存中请求的交易历史的限制
   CheckStartDateInTradeHistory();
  }
//+------------------------------------------------------------------+
//|  更改请求交易历史的开始日期                                      |
//+------------------------------------------------------------------+
void CheckStartDateInTradeHistory()
  {
//--- 初始时间间隔，如果我们现在开始工作
   datetime curr_start=TimeCurrent()-days*PeriodSeconds(PERIOD_D1);
//--- 确保交易历史的开始限制没有消失
//--- 预定日期超过1天
   if(curr_start-start>PeriodSeconds(PERIOD_D1))
      {
       //--- 修正在缓存区载入历史的开始日期
       start=curr_start;
       PrintFormat("New start limit of the trade history to be loaded: start => %s",
                   TimeToString(start));
       //--- 现在重新载入更新间隔的交易历史
       HistorySelect(start,end);
       //--- 为进一步比较，修正历史中交易和订单的计数器
       history_orders=HistoryOrdersTotal();
       deals=HistoryDealsTotal();
      }
  }
//+------------------------------------------------------------------+
/* Sample output:
  Limits of the history to be loaded: start - 2018.07.16 18:11, end - 2018.07.23 18:11
  The counters of orders, positions and deals are successfully initialized
  Number of orders has been changed. Previous value 0, current value 1
  Number of orders has been changed. Previous value 1, current value 0
  Number of positions has been changed. Previous value 0, current value 1
  Number of deals has been changed. Previous value 0, current value 1
```

© 2000-2025, MetaQuotes Ltd.

```
  Number of orders in the history has been changed. Previous value 0, current value 1
*/
```

另见

OrderSend、OrderSendAsync、OnTradeTransaction、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnTradeTransaction

当TradeTransaction事件发生时在EA中调用这个函数。这个函数旨在处理交易请求执行结果。

```
void  OnTradeTransaction()
   const MqlTradeTransaction&    trans,     // 交易事务结构
   const MqlTradeRequest&        request,   // 请求结构
   const MqlTradeResult&         result     // 回应结构
   );
```

参数

*trans*

[in] MqlTradeTransaction类型变量描述了在交易账户上进行的交易。

*request*

[in] MqlTradeRequest类型变量描述了导致交易的交易请求。它只包含TRADE_TRANSACTION_REQUEST类型交易的值。

*result*

[in] MqlTradeResult类型变量描述了导致交易的交易请求的执行结果。它只包含TRADE_TRANSACTION_REQUEST类型交易的值。

返回值

无返回值

注意

调用OnTradeTransaction()来处理在下列情况下由交易服务器发送至程序端的TradeTransaction事件：

- 使用OrderSend()/OrderSendAsync()函数，从MQL5程序发送交易请求及其后续执行；
- 通过GUI手动发送交易请求及其后续执行；
- 激活服务器上的挂单和止损单；
- 在交易服务器端执行操作。

关于交易类型的数据包含在*trans*变量的*类型*字段中。交易事务类型在ENUM_TRADE_TRANSACTION_TYPE枚举中描述：

- TRADE_TRANSACTION_ORDER_ADD -添加一个新的活跃订单
- TRADE_TRANSACTION_ORDER_UPDATE -更改现有订单
- TRADE_TRANSACTION_ORDER_DELETE -从活跃订单列表删除一个订单
- TRADE_TRANSACTION_DEAL_ADD -在历史中添加一笔交易
- TRADE_TRANSACTION_DEAL_UPDATE -在历史中改变一笔交易
- TRADE_TRANSACTION_DEAL_DELETE -从历史删除一笔交易
- TRADE_TRANSACTION_HISTORY_ADD -在历史中添加一个订单作为执行或取消的结果
- TRADE_TRANSACTION_HISTORY_UPDATE -在订单历史中更改一个订单
- TRADE_TRANSACTION_HISTORY_DELETE -从订单历史中删除一个订单
- TRADE_TRANSACTION_POSITION -与交易执行无关的持仓变化
- TRADE_TRANSACTION_REQUEST -已收到服务器处理交易请求的通知和处理结果。

© 2000-2025, MetaQuotes Ltd.

当处理TRADE_TRANSACTION_REQUEST类型交易时，需要分析OnTradeTransaction()函数的第二和第三个参数 -*request*和*result* -来获取额外信息。

发送买入交易请求会导致交易账户上的一系列交易：1）请求被接受进行处理，2）为该账户创建了一个适当的购买订单，3）然后执行订单，4）已执行订单从活跃订单列表中移除，5）添加到订单历史中，6）随后事务被添加到历史中以及 7）创建新持仓。所有这些阶段都是交易事务过程。每一个这类交易事务到达程序端就是TradeTransaction事件。不保证这些交易到达程序端的优先级。因此，在开发您的EA交易时，您不应该期望一组交易将在另一组之后到达。

当交易由EA的OnTradeTransaction()处理程序进行处理时，程序端将继续处理传入的交易事务。因此，在OnTradeTransaction()操作过程中，交易账户状态可能会发生变化。例如，当MQL5程序处理添加新订单时，它可以被执行，被从已开设订单的列表中删除以及被移动到历史中。所有这些事件的消息都将通知该程序。

交易事务队列长度由1024个要素组成。如果OnTradeTransaction()处理另一个事务时间太长，那么之前的事务就可能被队列中的新事物所取代。

适当调用OnTradeTransaction()之后调用OnTrade()处理程序。通常，在OnTrade()和OnTradeTransaction()调用的数量上没有确切的相关性。OnTrade()一次调用对应一次或多次的OnTradeTransaction调用。

每一个Trade事件都可能作为一个或多个交易请求的结果出现。交易请求使用OrderSend()或OrderSendAsync()，被发送到服务器。每个请求都可能导致多个交易事件。您不能依赖“一个请求，一个交易事件”的声明，因为事件的处理可能分为几个阶段执行，并且每个操作都可能更改订单的状态、持仓和交易历史记录。

EA样例，OnTradeTransaction()处理程序

```
//+------------------------------------------------------------------+
//|                                    OnTradeTransaction_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample listener of TradeTransaction events"
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   PrintFormat("LAST PING=%.f ms",
               TerminalInfoInteger(TERMINAL_PING_LAST)/1000.);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//---

  }
//+------------------------------------------------------------------+
//| TradeTransaction函数                                             |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//---
   static int counter=0;   // counter of OnTradeTransaction() calls
   static uint lasttime=0; // time of the OnTradeTransaction() last call
//---
   uint time=GetTickCount();
//--- 如果最后一个交易事务在1秒之前执行,
   if(time-lasttime>1000)
     {
      counter=0; // 那么这是一个新交易操作,计数器可被重置
      if(IS_DEBUG_MODE)
         Print(" New trade operation");
     }
   lasttime=time;
   counter++;
   Print(counter,". ",__FUNCTION__);
//--- 交易请求执行的结果
   ulong            lastOrderID   =trans.order;
   ENUM_ORDER_TYPE  lastOrderType =trans.order_type;
   ENUM_ORDER_STATE lastOrderState=trans.order_state;
//--- 执行交易的交易品种名称
   string trans_symbol=trans.symbol;
//--- 交易事务类型
   ENUM_TRADE_TRANSACTION_TYPE  trans_type=trans.type;
   switch(trans.type)
     {
      case  TRADE_TRANSACTION_POSITION:   //持仓变化
        {
         ulong pos_ID=trans.position;
         PrintFormat("MqlTradeTransaction: Position  #%I64u %s modified: SL=%.5f TP=%.
                     pos_ID,trans_symbol,trans.price_sl,trans.price_tp);
        }
      break;
      case TRADE_TRANSACTION_REQUEST:     // sending a trade request
         PrintFormat("MqlTradeTransaction: TRADE_TRANSACTION_REQUEST");
         break;
      case TRADE_TRANSACTION_DEAL_ADD:    // 添加一个交易
        {
         ulong          lastDealID   =trans.deal;
```

© 2000-2025, MetaQuotes Ltd.

```
         ENUM_DEAL_TYPE lastDealType =trans.deal_type;
         double         lastDealVolume=trans.volume;
         //--- 内部系统中的交易ID(Trade ID) - 由交易所分配的单号
         string Exchange_ticket="";
         if(HistoryDealSelect(lastDealID))
            Exchange_ticket=HistoryDealGetString(lastDealID,DEAL_EXTERNAL_ID);
         if(Exchange_ticket!="")
            Exchange_ticket=StringFormat("(Exchange deal=%s)",Exchange_ticket);

         PrintFormat("MqlTradeTransaction: %s deal #%I64u %s %s %.2f lot   %s",EnumToS
                     lastDealID,EnumToString(lastDealType),trans_symbol,lastDealVolume
        }
      break;
      case TRADE_TRANSACTION_HISTORY_ADD: // 在历史中添加一个订单
        {
         //--- 内部系统中的订单ID - 由交易所分配的单号
         string Exchange_ticket="";
         if(lastOrderState==ORDER_STATE_FILLED)
           {
            if(HistoryOrderSelect(lastOrderID))
               Exchange_ticket=HistoryOrderGetString(lastOrderID,ORDER_EXTERNAL_ID);
            if(Exchange_ticket!="")
               Exchange_ticket=StringFormat("(Exchange ticket=%s)",Exchange_ticket);
           }
         PrintFormat("MqlTradeTransaction: %s order #%I64u %s %s %s   %s",EnumToStrinq
                     lastOrderID,EnumToString(lastOrderType),trans_symbol,EnumToStrinq
        }
      break;
      default: // 其他交易事务
        {
         //--- 内部系统中的订单ID - 由莫斯科交易所(Moscow Exchange)分配的单号
         string Exchange_ticket="";
         if(lastOrderState==ORDER_STATE_PLACED)
           {
            if(OrderSelect(lastOrderID))
               Exchange_ticket=OrderGetString(ORDER_EXTERNAL_ID);
            if(Exchange_ticket!="")
               Exchange_ticket=StringFormat("Exchange ticket=%s",Exchange_ticket);
           }
         PrintFormat("MqlTradeTransaction: %s order #%I64u %s %s   %s",EnumToString(tr
                     lastOrderID,EnumToString(lastOrderType),EnumToString(lastOrderSta
        }
      break;
     }
//--- 订单单号
   ulong orderID_result=result.order;
   string retcode_result=GetRetcodeID(result.retcode);
   if(orderID_result!=0)
      PrintFormat("MqlTradeResult: order #%d retcode=%s ",orderID_result,retcode_resul
```

© 2000-2025, MetaQuotes Ltd.

```
//---
  }
//+------------------------------------------------------------------+
//| 将数字响应代码转换为字符串记忆                                   |
//+------------------------------------------------------------------+
string GetRetcodeID(int retcode)
  {
   switch(retcode)
     {
      case 10004: return("TRADE_RETCODE_REQUOTE");             break;
      case 10006: return("TRADE_RETCODE_REJECT");              break;
      case 10007: return("TRADE_RETCODE_CANCEL");              break;
      case 10008: return("TRADE_RETCODE_PLACED");              break;
      case 10009: return("TRADE_RETCODE_DONE");                break;
      case 10010: return("TRADE_RETCODE_DONE_PARTIAL");        break;
      case 10011: return("TRADE_RETCODE_ERROR");               break;
      case 10012: return("TRADE_RETCODE_TIMEOUT");             break;
      case 10013: return("TRADE_RETCODE_INVALID");             break;
      case 10014: return("TRADE_RETCODE_INVALID_VOLUME");      break;
      case 10015: return("TRADE_RETCODE_INVALID_PRICE");       break;
      case 10016: return("TRADE_RETCODE_INVALID_STOPS");       break;
      case 10017: return("TRADE_RETCODE_TRADE_DISABLED");      break;
      case 10018: return("TRADE_RETCODE_MARKET_CLOSED");       break;
      case 10019: return("TRADE_RETCODE_NO_MONEY");            break;
      case 10020: return("TRADE_RETCODE_PRICE_CHANGED");       break;
      case 10021: return("TRADE_RETCODE_PRICE_OFF");           break;
      case 10022: return("TRADE_RETCODE_INVALID_EXPIRATION");  break;
      case 10023: return("TRADE_RETCODE_ORDER_CHANGED");       break;
      case 10024: return("TRADE_RETCODE_TOO_MANY_REQUESTS");   break;
      case 10025: return("TRADE_RETCODE_NO_CHANGES");          break;
      case 10026: return("TRADE_RETCODE_SERVER_DISABLES_AT");  break;
      case 10027: return("TRADE_RETCODE_CLIENT_DISABLES_AT");  break;
      case 10028: return("TRADE_RETCODE_LOCKED");              break;
      case 10029: return("TRADE_RETCODE_FROZEN");              break;
      case 10030: return("TRADE_RETCODE_INVALID_FILL");        break;
      case 10031: return("TRADE_RETCODE_CONNECTION");          break;
      case 10032: return("TRADE_RETCODE_ONLY_REAL");           break;
      case 10033: return("TRADE_RETCODE_LIMIT_ORDERS");        break;
      case 10034: return("TRADE_RETCODE_LIMIT_VOLUME");        break;
      case 10035: return("TRADE_RETCODE_INVALID_ORDER");       break;
      case 10036: return("TRADE_RETCODE_POSITION_CLOSED");     break;
      default:
         return("TRADE_RETCODE_UNKNOWN="+IntegerToString(retcode));
         break;
     }
//---
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

OrderSend、OrderSendAsync、OnTradeTransaction、交易请求结构、交易事务结构、交易事务类型、交易操作类型、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnBookEvent

当BookEvent事件发生时在指标和EA中调用这个函数。它是用来处理市场深度的变化。

```
void  OnBookEvent(
   const string&  symbol          //交易品种
   );
```

参数

*交易品种*

[in] BookEvent到达的交易品种名称

返回值

无返回值

注意

若要获得任何交易品种的BookEvent事件，只需使用MarketBookAdd()函数，为该交易品种订阅接收该事件。若要为特定交易品种取消订阅接收BookEvent，则调用MarketBookRelease()函数。

BookEvent在整个图表中进行传播。这意味着，如果图表上的一个应用程序使用MarketBookAdd函数订阅BookEvent，那么在相同图表上启用的所有其他指标和EA以及OnBookEvent()处理程序也会收到这个事件。因此，有必要分析传递到OnBookEvent()处理程序的交易品种名称，并将其作为*交易品种*参数。

为运行在相同图表上的所有应用程序提供按交易品种分类的单独的BookEvent计数器。这意味着每个图表对不同的交易品种可能有多个订阅，并且为每个交易品种提供一个计数器。从BookEvent订阅和取消订阅，只在一个图表中更改指定交易品种的订阅计数器。换句话说，对于相同的交易品种，BookEvent可能有两个相邻的图表，但订阅计数值不同。

初始的订阅计数值为零。每调用一次MarketBookAdd()，图表上特定交易品种的订阅计数器都会增加1个（图表交易品种和MarketBookAdd()中的交易品种不必匹配）。当调用MarketBookRelease()时，在图表中指定交易品种的订阅计数器就会减少1个。任何交易品种的BookEvent事件都在图表中广播，直至计数器的值等于零。因此，很重要的一点就是，每个包含了MarketBookAdd()调用的MQL5程序都在其工作结束时使用MarketBookRelease()为每个交易品种正确取消订阅获取事件。为此，MarketBookAdd()和MarketBookRelease()的调用数量甚至都应该适用于整个MQL5程序生命周期内的每个调用。在程序中使用标识或自定义订阅计数器可以使您安全地处理BookEvent事件，并阻止禁用订阅，以在同一图表内的第三方程序中获取该事件。

即使处理之前的BookEvent处理还没有结束，BookEvent 事件也不会被跳过，会始终被置于队列之中。

例如

```
//+------------------------------------------------------------------+
//|                                          OnBookEvent_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
```

© 2000-2025, MetaQuotes Ltd.

```
#property link      "https://www.mql5.com/en/articles/2635"
#property version   "1.00"
#property description "Example of measuring the market depth refresh rate using OnBook
#property description "The code is taken from the article https://www.mql5.com/en/arti
//--- 输入参数
input ulong ExtCollectTime   =30;  // test time in seconds
input ulong ExtSkipFirstTicks=10;  // number of ticks skipped at start
//--- 订阅BookEvent事件的标识
bool book_subscribed=false;
//--- 接受来自市场深度的请求的数组
MqlBookInfo  book[];
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 显示开始
   Comment(StringFormat("Waiting for the first %I64u ticks to arrive",ExtSkipFirstTick
   PrintFormat("Waiting for the first %I64u ticks to arrive",ExtSkipFirstTicks);
//--- 启用市场深度广播
   if(MarketBookAdd(_Symbol))
     {
      book_subscribed=true;
      PrintFormat("%s: MarketBookAdd(%s) function returned true",__FUNCTION__,_Symbol)
     }
   else
      PrintFormat("%s: MarketBookAdd(%s) function returned false! GetLastError()=%d",_
//--- 成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 去初始化EA交易                                                   |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 显示去初始化原因代码
   Print(__FUNCTION__,": Deinitialization reason code = ",reason);
//--- 取消订阅以获取市场深度事件
   if(book_subscribed)
     {
      if(!MarketBookRelease(_Symbol))
         PrintFormat("%s: MarketBookRelease(%s) returned false! GetLastError()=%d",_Sy
      else
         book_subscribed=false;
     }
//---
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| BookEvent函数                                                    |
//+------------------------------------------------------------------+
void OnBookEvent(const string &symbol)
  {
   static ulong starttime=0;              // 测试开始时间
   static ulong tickcounter=0;            // 市场深度更新计数器
//--- 只有当我们自己订阅市场深度事件时处理它们
   if(!book_subscribed)
      return;
//--- 只为特定交易品种计算更新
   if(symbol!=_Symbol)
      return;
//--- 跳过第一个报价来清空队列和进行准备
   tickcounter++;
   if(tickcounter<ExtSkipFirstTicks)
      return;
//--- 记住开始时间
   if(tickcounter==ExtSkipFirstTicks)
      starttime=GetMicrosecondCount();
//--- 请求市场深度数据
   MarketBookGet(symbol,book);
//--- 什么时候停止?
   ulong endtime=GetMicrosecondCount()-starttime;
   ulong ticks  =1+tickcounter-ExtSkipFirstTicks;
// 从测试开始以后，已经过去多少毫秒?
   if(endtime>ExtCollectTime*1000*1000)
     {
      PrintFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endtime/1000.0
      ExpertRemove();
      return;
     }
//--- 在评论字段中显示计数器
   if(endtime>0)
      Comment(StringFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endti
  }
```

另见

MarketBookAdd、MarketBookRelease、MarketBookGet、OnTrade、OnTradeTransaction、OnTick、事件处理函数、程序运行、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnChartEvent

当ChartEvent事件发生时在指标和EA中调用这个函数。这个函数意在处理由用户或者MQL5程序所做的图表更改。

```
void  OnChartEvent()
   const int       id,       // 事件ID
   const long&     lparam,   // 长整型(long)事件参数
   const double&   dparam,   // 长整型(double)事件参数
   const string&   sparam    // 字符串型(string)事件参数
   );
```

参数

*id*

[in] 来自ENUM_CHART_EVENT枚举的事件ID。

*lparam*

[in] 长整型(long)事件参数

*dparam*

[in] 双精度型(double)事件参数

*sparam*

[in] 字符串型(string)事件参数

返回值

无返回值

注意

共有11种事件类型可以使用预定义OnChartEvent()函数来处理。为自定义事件提供了从CHARTEVENT_CUSTOM到CHARTEVENT_CUSTOM_LAST包含的65535个ID。若要生成自定义事件，请使用EventChartCustom()函数。

来自ENUM_CHART_EVENT 枚举的简短事件描述：

- CHARTEVENT_KEYDOWN —当图表窗口处于焦点时，按下键盘上的一个按键；
- CHARTEVENT_MOUSE_MOVE — 移动鼠标，并且鼠标按键点击 (如果图表CHART_EVENT_MOUSE_MOVE=true)；
- CHARTEVENT_OBJECT_CREATE — 创建一个图形对象 (如果图表CHART_EVENT_OBJECT_CREATE=true)；
- CHARTEVENT_OBJECT_CHANGE —通过属性对话框更改对象属性；
- CHARTEVENT_OBJECT_DELETE — 删除一个图形对象 (如果图表CHART_EVENT_OBJECT_DELETE=true)；
- CHARTEVENT_CLICK —点击图表；
- CHARTEVENT_OBJECT_CLICK —鼠标单击属于图表的图形对象；
- CHARTEVENT_OBJECT_DRAG —用鼠标拖动图形对象；
- CHARTEVENT_OBJECT_ENDEDIT —在图形对象的“编辑”输入框中完成文本编辑 (OBJ_EDIT)；
- CHARTEVENT_CHART_CHANGE —更改图表；
- CHARTEVENT_CUSTOM+n — 自定义事件ID，这里n的范围是从0到65535。CHARTEVENT_CUSTOM_LAST包含最后可接受的自定义事件ID (CHARTEVENT_CUSTOM+65535)。

© 2000-2025, MetaQuotes Ltd.

所有MQL5程序都使用了不同于应用程序主线程的其他线程。主应用程序线程负责处理所有的Windows系统消息，反过来，作为该处理的结果，为其自身的应用程序生成Windows消息。例如，在图表上移动鼠标（WM_MOUSE_MOVE事件）为随后呈现的应用程序窗口生成几个系统消息，并将内部消息发送到在图表上启动的EA和指标。可能会出现这样的情况，主要的应用程序线程还没有处理WM_PAINT系统消息（因此还没有呈现修改过的图表），而EA或指标却已经收到了鼠标移动事件。在这种情况下，只有在呈现图表之后，图表属性CHART_FIRST_VISIBLE_BAR才会更改。

对于每种事件类型，OnChartEvent()函数的输入都具有处理该事件所需的某些值。该表列出了通过参数传递的事件和值。

| 事件 | 'id'参数值 | 'lparam'参数值 | 'dparam'参数值 | 'sparam'参数值 |
|---|---|---|---|---|
| Keystroke事件 | CHARTEVENT_KEYDOWN | 按下按键代码 | 在按键处于按下状态时生成的按键响应的数量 | 位掩码的字符串值，描述键盘按键的状态 |
| 鼠标事件（如果图表CHART_EVENT_MOUSE_MOVE=true） | CHARTEVENT_MOUSE_MOVE | X坐标 | Y坐标 | 位掩码的字符串值，描述鼠标按键的状态 |
| 鼠标滚轮事件（如果CHART_EVENT_MOUSE_WHEEL=true为图表设置） | CHARTEVENT_MOUSE_WHEEL | 键盘按键和鼠标按键的状态标识，光标的X和Y坐标。请参阅示例中的描述 | 鼠标滚轮滚动的Delta值 | — |
| 创建一个图形对象（如果图表CHART_EVENT_OBJECT_CREATE=true） | CHARTEVENT_OBJECT_CREATE | — | — | 已创建图形对象的名称 |
| 通过属性对话框更改对象属性 | CHARTEVENT_OBJECT_CHANGE | — | — | 已更改图形对象的名称 |
| 移除一个图形对象（如果图表CHART_EVENT_OBJECT_DELETE=true） | CHARTEVENT_OBJECT_DELETE | — | — | 已移除图形对象的名称 |
| 图表点击图表 | CHARTEVENT_CLICK | X坐标 | Y坐标 | — |
| 鼠标点击图形对象 | CHARTEVENT_OBJECT_CLICK | X坐标 | Y坐标 | 事件所发生的图形对象的名称 |

© 2000-2025, MetaQuotes Ltd.

| 事件 | 'id'参数值 | 'lparam'参数值 | 'dparam'参数值 | 'sparam'参数值 |
|---|---|---|---|---|
| 使用鼠标移动图形对象 | CHARTEVENT_OBJECT_DRAG | — | — | 已移动图形对象的名称 |
| 在图形对象输入框的"输入字段"中完成文本编辑 | CHARTEVENT_OBJECT_ENDEDIT | — | — | "输入字段"图形对象的名称，在这里完成文本编辑 |
| 通过属性对话框窗口，调整图表大小或更改图表属性 | CHARTEVENT_CHART_CHANGE | — | — | — |
| N个数字的自定义事件 | CHARTEVENT_CUSTOM+N | 通过EventChartCustom()函数定义的值 | 通过EventChartCustom()函数定义的值 | 通过EventChartCustom()函数定义的值 |

示例图表事件监听器：

```
//+------------------------------------------------------------------+
//|                                          OnChartEvent_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample chart event listener and custom events generator"
//--- 服务键ID
#define KEY_NUMPAD_5        12
#define KEY_LEFT            37
#define KEY_UP              38
#define KEY_RIGHT           39
#define KEY_DOWN            40
#define KEY_NUMLOCK_DOWN    98
#define KEY_NUMLOCK_LEFT   100
#define KEY_NUMLOCK_5      101
#define KEY_NUMLOCK_RIGHT  102
#define KEY_NUMLOCK_UP     104
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 显示CHARTEVENT_CUSTOM常数值
   Print("CHARTEVENT_CUSTOM=",CHARTEVENT_CUSTOM);
//---
   Print("Launched the EA ",MQLInfoString(MQL5_PROGRAM_NAME));
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置接收图表对象创建事件的标识
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- 设置接收图表对象移除事件的标识
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//--- 启用鼠标滚轮来滚动信息
   ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
//--- 强制更新图表属性确保做好事件处理的准备
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 用于生成自定义事件的报价计数器
   static int tick_counter=0;
//--- 通过这个值除以累积的报价
   int simple_number=113;
//---
   tick_counter++;
//--- 如果报价计数器是simple_number的倍数，则发送自定义事件
   if(tick_counter%simple_number==0)
     {
      //--- 形成自定义事件ID 从0到65535）
      ushort custom_event_id=ushort(tick_counter%65535);
      //---  发送参数填充的自定义事件
      EventChartCustom(ChartID(),custom_event_id,tick_counter,SymbolInfoDouble(Symbol
      //--- 添加到日志来分析示例结果
      Print(__FUNCTION__,": Sent a custom event ID=",custom_event_id);
     }
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 按键响应
   if(id==CHARTEVENT_KEYDOWN)
     {
      switch((int)lparam)
        {
         case KEY_NUMLOCK_LEFT:  Print("Pressed KEY_NUMLOCK_LEFT");   break;
         case KEY_LEFT:          Print("Pressed KEY_LEFT");           break;
```

© 2000-2025, MetaQuotes Ltd.

```
         case KEY_NUMLOCK_UP:    Print("Pressed KEY_NUMLOCK_UP");     break;
         case KEY_UP:            Print("Pressed KEY_UP");             break;
         case KEY_NUMLOCK_RIGHT: Print("Pressed KEY_NUMLOCK_RIGHT"); break;
         case KEY_RIGHT:         Print("Pressed KEY_RIGHT");          break;
         case KEY_NUMLOCK_DOWN:  Print("Pressed KEY_NUMLOCK_DOWN");   break;
         case KEY_DOWN:          Print("Pressed KEY_DOWN");           break;
         case KEY_NUMPAD_5:      Print("Pressed KEY_NUMPAD_5");       break;
         case KEY_NUMLOCK_5:     Print("Pressed KEY_NUMLOCK_5");      break;
         default:                Print("Pressed unlisted key");
        }
     }
//--- 左击图表
   if(id==CHARTEVENT_CLICK)
      Print("Mouse click coordinates on a chart: x = ",lparam,"  y = ",dparam);
//--- 点击图形对象
   if(id==CHARTEVENT_OBJECT_CLICK)
      Print("Clicking a mouse button on an object named '"+sparam+"'");
//--- 被移除的对象
   if(id==CHARTEVENT_OBJECT_DELETE)
      Print("Removed object named ",sparam);
//--- 被创建的对象
   if(id==CHARTEVENT_OBJECT_CREATE)
      Print("Created object named ",sparam);
//--- 被更改的对象
   if(id==CHARTEVENT_OBJECT_CHANGE)
      Print("Changed object named ",sparam);
//--- 被移动的对象或被改变的定位点坐标
   if(id==CHARTEVENT_OBJECT_DRAG)
      Print("Changing anchor points of object named ",sparam);
//--- 在 "编辑"图形对象的输入字段更改文本
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
      Print("Changed text in Edit object ",sparam,"  id=",id);
//--- 鼠标移动事件
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
   if(id==CHARTEVENT_MOUSE_WHEEL)
     {
      //--- 考虑这个事件的鼠标按键和滚轮的状态
      int flg_keys = (int)(lparam>>32);          // Ctrl 和 Shift 键，鼠标按键的状态标识
      int x_cursor = (int)(short)lparam;         // 鼠标滚轮事件发生的X坐标
      int y_cursor = (int)(short)(lparam>>16);   // 鼠标滚轮事件发生的Y坐标
      int delta    = (int)dparam;                // 鼠标滚动的总值，到达 +120 或 -120 时触
      //--- 处理标识
      string str_keys="";
      if((flg_keys&0x0001)!=0)
         str_keys+="LMOUSE ";
      if((flg_keys&0x0002)!=0)
         str_keys+="RMOUSE ";
      if((flg_keys&0x0004)!=0)
```

© 2000-2025, MetaQuotes Ltd.

```
         str_keys+="SHIFT ";
      if((flg_keys&0x0008)!=0)
         str_keys+="CTRL ";
      if((flg_keys&0x0010)!=0)
         str_keys+="MMOUSE ";
      if((flg_keys&0x0020)!=0)
         str_keys+="X1MOUSE ";
      if((flg_keys&0x0040)!=0)
         str_keys+="X2MOUSE ";

      if(str_keys!="")
         str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1)+"'";
      PrintFormat("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x,
     }
//--- 使用属性对话框窗口调整图表大小或更改图表属性的事件
   if(id==CHARTEVENT_CHART_CHANGE)
      Print("Changing the chart size or properties");
//--- 自定义事件
   if(id>CHARTEVENT_CUSTOM)
      PrintFormat("Custom event ID=%d, lparam=%d, dparam=%G, sparam=%s",id,lparam,dpar
  }
//+------------------------------------------------------------------+
//| 鼠标点选位置                                                     |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // 鼠标左移
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // 鼠标右移
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // 鼠标居中
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // 鼠标点选第一个X键
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // 鼠标点选第二个 X 键
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // 移位键
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // 控制键
   return(res);
  }
```

另见

EventChartCustom、图表事件类型、事件处理函数、程序运行、客户端事件

© 2000-2025, MetaQuotes Ltd.

# OnTester

当Tester事件发生时在EA中调用这个函数，测试之后执行必要的操作。

```
double  OnTester(void);
```

返回值

用于评估测试结果的自定义标准优化的值。

注意

OnTester()函数只有在测试EA时可被使用，主要用于计算在优化输入参数时用作'Custom max（最大自定义）'准则的值。

在遗传优化过程中，在一代内对结果进行降序排列。这意味着从优化准则的角度来看，具有最高价值的结果被认为是最高的。这种排序最糟糕的值被排在最后，并随后丢弃。因此，它们不参与形成下一代。

因此，OnTester()函数不仅允许您创建和保存您自己的测试结果报告，还可以控制优化过程，找出交易策略的最佳参数。

以下就是计算自定义准则优化的示例。意在计算结余图形的线性回归。这在使用结余图形优化策略，并将结果与"结余 + 最大夏普比率"准则进行比较文章中进行描述。

```
//+------------------------------------------------------------------+
//|                                                OnTester_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA with the OnTester() handler"
#property description "As a custom optimization criterion, "
#property description "the ratio of the balance graph linear regression"
#property description "divided by the deviation mean-square error is returned"
//--- 包含交易操作类
#include <Trade\Trade.mqh>
//--- EA输入参数
input double Lots                = 0.1;      // 交易量
input int    Slippage            = 10;       // 可允许滑移
input int    MovingPeriod        = 80;       // 移动平均线时间
input int    MovingShift         = 6;        // 移动平均线移位
//--- 全局变量
int    IndicatorHandle=0;  // 指标句柄
bool   IsHedging=false;    // 账户标识
CTrade trade;              // 为执行交易操作
//---
#define EA_MAGIC 18052018
//+------------------------------------------------------------------+
//| 检查持仓开仓条件                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void CheckForOpen(void)
  {
   MqlRates rt[2];
//--- 仅在新柱形图开始进行交易
   if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
     {
      Print("CopyRates of ",_Symbol," failed, no history");
      return;
     }
//--- 报价量
   if(rt[1].tick_volume>1)
      return;
//--- 接收移动平均线值
   double   ma[1];
   if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
     {
      Print("CopyBuffer from iMA failed, no data");
      return;
     }
//--- 检查信号是否存在
   ENUM_ORDER_TYPE signal=WRONG_VALUE;
//--- 蜡烛图开盘高于移动平均线但收盘低于移动平均线
   if(rt[0].open>ma[0] && rt[0].close<ma[0])
      signal=ORDER_TYPE_BUY;     //买入信号
   else // 蜡烛图开盘低于移动平均线但收盘高于移动平均线
     {
      if(rt[0].open<ma[0] && rt[0].close>ma[0])
         signal=ORDER_TYPE_SELL;// 卖出信号
     }
//--- 附加检查
   if(signal!=WRONG_VALUE)
     {
      if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
        {
         double price=SymbolInfoDouble(_Symbol,signal==ORDER_TYPE_SELL ? SYMBOL_BID:SY
         trade.PositionOpen(_Symbol,signal,Lots,price,0,0);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| 检查持仓平仓条件                                                 |
//+------------------------------------------------------------------+
void CheckForClose(void)
  {
   MqlRates rt[2];
//--- 仅在新柱形图开始进行交易
   if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("CopyRates of ",_Symbol," failed, no history");
       return;
      }
   if(rt[1].tick_volume>1)
      return;
//--- 接收移动平均线值
   double   ma[1];
   if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
     {
      Print("CopyBuffer from iMA failed, no data");
      return;
     }
//--- 已使用PositionSelect()提前选择持仓
   bool signal=false;
   long type=PositionGetInteger(POSITION_TYPE);
//--- 蜡烛图开盘高于移动平均线但收盘低于移动平均线—关闭卖出持仓
   if(type==(long)POSITION_TYPE_SELL && rt[0].open>ma[0] && rt[0].close<ma[0])
      signal=true;
//--- 蜡烛图开盘低于移动平均线但收盘高于移动平均线—关闭买入持仓
   if(type==(long)POSITION_TYPE_BUY && rt[0].open<ma[0] && rt[0].close>ma[0])
      signal=true;
//--- 附加检查
   if(signal)
     {
      if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
         trade.PositionClose(_Symbol,Slippage);
     }
//---
  }
//+------------------------------------------------------------------+
//| 选择一个考虑账户类型的持仓:单边持仓或锁仓持仓                                  |
//+------------------------------------------------------------------+
bool SelectPosition()
  {
   bool res=false;
//--- 选择一个锁仓账户系统的持仓
   if(IsHedging)
     {
      uint total=PositionsTotal();
      for(uint i=0; i<total; i++)
        {
         string position_symbol=PositionGetSymbol(i);
         if(_Symbol==position_symbol && EA_MAGIC==PositionGetInteger(POSITION_MAGIC))
           {
            res=true;
            break;
           }
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 选择一个单边账户系统的持仓
   else
     {
      if(!PositionSelect(_Symbol))
         return(false);
      else
         return(PositionGetInteger(POSITION_MAGIC)==EA_MAGIC); //---check Magic numbe
      }
//--- 执行结果
   return(res);
  }
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit(void)
  {
//--- 设置一个交易类型:单边交易类型或锁仓交易类型
   IsHedging=((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)==ACCOU
//--- 初始化一个用于正确持仓控制的对象
   trade.SetExpertMagicNumber(EA_MAGIC);
   trade.SetMarginMode();
   trade.SetTypeFillingBySymbol(Symbol());
   trade.SetDeviationInPoints(Slippage);
//--- 创建移动平均线指标
   IndicatorHandle=iMA(_Symbol,_Period,MovingPeriod,MovingShift,MODE_SMA,PRICE_CLOSE);
   if(IndicatorHandle==INVALID_HANDLE)
     {
      printf("Error creating iMA indicator");
      return(INIT_FAILED);
     }
//--- ok
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick(void)
  {
//--- 如果已开仓,请检查平仓条件
   if(SelectPosition())
      CheckForClose();
// 检查持仓开仓条件
   CheckForOpen();
//---
  }
//+------------------------------------------------------------------+
//| 测试函数                                                         |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
double OnTester()
  {
//--- 自定义准则优化值 越高越好）
   double ret=0.0;
//--- 将交易结果转到数组
   double array[];
   double trades_volume;
   GetTradeResultsToArray(array,trades_volume);
   int trades=ArraySize(array);
//--- 如果少于10个交易，那么测试结果没有正值
   if(trades<10)
      return (0);
//--- 每个交易的平均结果
   double average_pl=0;
   for(int i=0;i<ArraySize(array);i++)
      average_pl+=array[i];
   average_pl/=trades;
//--- 显示单测试模式的消息
   if(MQLInfoInteger(MQL_TESTER) && !MQLInfoInteger(MQL_OPTIMIZATION))
      PrintFormat("%s: Trades=%d, Average profit=%.2f",__FUNCTION__,trades,average_pl);
//--- 计算利润图的线性回归比
   double a,b,std_error;
   double chart[];
   if(!CalculateLinearRegression(array,chart,a,b))
      return (0);
//--- 计算回归线与图表偏差的误差
   if(!CalculateStdError(chart,a,b,std_error))
      return (0);
//--- 计算趋势利润与标准偏差的比率
   ret=(std_error == 0.0) ? a*trades : a*trades/std_error;
//--- 返回自定义标准优化值
   return(ret);
  }
//+------------------------------------------------------------------+
//| 获取交易的利润/亏损的数组                                          |
//+------------------------------------------------------------------+
bool GetTradeResultsToArray(double &pl_results[],double &volume)
  {
//--- 请求完整的交易历史
   if(!HistorySelect(0,TimeCurrent()))
      return (false);
   uint total_deals=HistoryDealsTotal();
   volume=0;
//--- 设置预付款数组的初始大小 - 根据历史中的交易数量
   ArrayResize(pl_results,total_deals);
//--- 解决交易结果的交易计数器 - 盈利或亏损
   int counter=0;
   ulong ticket_history_deal=0;
//--- 检查所有交易
```

© 2000-2025, MetaQuotes Ltd.

```
   for(uint i=0;i<total_deals;i++)
     {
      //--- 选择一个交易
      if((ticket_history_deal=HistoryDealGetTicket(i))>0)
        {
         ENUM_DEAL_ENTRY deal_entry  =(ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket_hi
         long            deal_type   =HistoryDealGetInteger(ticket_history_deal,DEAL_T
         double          deal_profit =HistoryDealGetDouble(ticket_history_deal,DEAL_PR
         double          deal_volume =HistoryDealGetDouble(ticket_history_deal,DEAL_VO
         //--- 我们只对交易操作感兴趣
         if((deal_type!=DEAL_TYPE_BUY) && (deal_type!=DEAL_TYPE_SELL))
            continue;
         //--- 只有解决盈利/亏损的交易
         if(deal_entry!=DEAL_ENTRY_IN)
           {
            //--- 将交易结果写入数组并增加交易的计数器
            pl_results[counter]=deal_profit;
            volume+=deal_volume;
            counter++;
           }
        }
     }
//--- 设置数组的最终大小
   ArrayResize(pl_results,counter);
   return (true);
  }
//+------------------------------------------------------------------+
//| 计算线性回归 y=a*x+b                                             |
//+------------------------------------------------------------------+
bool CalculateLinearRegression(double  &change[],double &chartline[],
                               double  &a_coef,double  &b_coef)
  {
//--- 检查数据充分性
   if(ArraySize(change)<3)
      return (false);
//--- 创建一个累积图表数组
   int N=ArraySize(change);
   ArrayResize(chartline,N);
   chartline[0]=change[0];
   for(int i=1;i<N;i++)
      chartline[i]=chartline[i-1]+change[i];
//--- 现在，计算回归比率
   double x=0,y=0,x2=0,xy=0;
   for(int i=0;i<N;i++)
     {
      x=x+i;
      y=y+chartline[i];
      xy=xy+i*chartline[i];
      x2=x2+i*i;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   a_coef=(N*xy-x*y)/(N*x2-x*x);
   b_coef=(y-a_coef*x)/N;
//---
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算指定a和b的平均-平方偏差误差                                 |
//+------------------------------------------------------------------+
bool  CalculateStdError(double  &data[],double  a_coef,double  b_coef,double &std_err)
  {
//--- 平方和错误
   double error=0;
   int N=ArraySize(data);
   if(N<=2)
      return (false);
   for(int i=0;i<N;i++)
      error+=MathPow(a_coef*i+b_coef-data[i],2);
   std_err=MathSqrt(error/(N-2));
//---
   return (true);
  }
```

另见

测试交易策略、TesterHideIndicators、处理优化结果、TesterStatistics、OnTesterInit、OnTesterDeinit、OnTesterPass、MQL_TESTER、MQL_OPTIMIZATION、FileOpen、FileWrite、FileLoad、FileSave

© 2000-2025, MetaQuotes Ltd.

# OnTesterInit

当TesterInit事件发生时在EA中调用这个函数，以便在策略测试优化之前执行必要的操作。有两种函数类型。

返回结果的版本

```
int  OnTesterInit(void);
```

返回值

int 类型值，0意味着优化开始之前图表上启动的EA的成功初始化。

建议使用返回执行结果的OnTesterInit()调用，因为它不仅可以程序初始化，还可以在早期优化停止的情况下返回一个错误代码。返回INIT_SUCCEEDED 以外的任何值，(0)表示错误，没有启动优化。

没有结果返回的版本只为与旧代码兼容而保留。不建议使用

```
void  OnTesterInit(void);
```

注意

TesterInit事件在策略测试开始期间EA优化之前生成。在该事件中，带有OnTesterDeInit()或OnTesterPass()事件处理程序的EA会在单独的程序端图表上自动下载。它包括测试中已经制定的交易品种和周期。

这类事件接收TesterInit、TesterDeinit和TesterPass事件，但不接收 Init、Deinit和NewTick事件。因此，优化期间处理每次传递结果的所有必要逻辑都应该在OnTesterInit ()、OnTesterDeinit ()和OnTesterPass ()处理程序中实施。

策略优化期间每次单独传递的结果都可以使用FrameAdd ()函数，通过来自OnTester ()处理程序的框架进行传递。

OnTesterInit()函数被用来在优化开始之前启动EA交易，以便进一步处理优化结果。它经常与OnTesterDeinit()处理程序一起使用。

OnTesterInit()的执行时间是有限的。如果超过限定时间，EA会被强制停止，而优化本身也会被取消。在测试日志中显示一条消息：

测试器 OnTesterInit工作时间过长。测试器不能被初始化。

来自OnTick的示例。添加OnTesterInit()处理程序来设置优化参数：

```
//+------------------------------------------------------------------+
//|                                          OnTesterInit_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA with the OnTesterInit() handler,"
#property description "in which values and limitations of "
#property description "inputs during optimization are set"
```

© 2000-2025, MetaQuotes Ltd.

```
input double lots=0. 1;        // 交易量手数
input double kATR=3;           // ATR中的信号蜡烛长度
input int    ATRperiod=20;     // ATR指标周期
input int    holdbars=8;       // 保持持仓的柱形图数量
input int    slippage=10;      // 可允许滑移
input bool   revers=false;     //反向信号?
input ulong  EXPERT_MAGIC=0;   // EA幻数
//--- 用于存储ATR指标句柄
int atr_handle;
//--- 在这里,我们将存储ATR最后值和蜡烛体
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
//--- 请记住优化开始时间
datetime optimization_start;
//--- 用于在优化结束后在图表上显示持续时间
string report;
//+------------------------------------------------------------------+
//| TesterInit 函数                                                  |
//+------------------------------------------------------------------+
void OnTesterInit()
  {
//--- 设置优化的输入值
   ParameterSetRange("lots",false,0.1,0,0,0);
   ParameterSetRange("kATR",true,3.0,1.0,0.3,7.0);
   ParameterSetRange("ATRperiod",true,10,15,1,30);
   ParameterSetRange("holdbars",true,5,3,1,15);
   ParameterSetRange("slippage",false,10,0,0,0);
   ParameterSetRange("revers",true,false,false,1,true);
   ParameterSetRange("EXPERT_MAGIC",false,123456,0,0,0);
   Print("Initial values and optimization parameter limitations are set");
//--- 记得开始优化
   optimization_start=TimeLocal();
   report=StringFormat("%s: optimization launched at %s",
                       __FUNCTION__,TimeToString(TimeLocal(),TIME_MINUTES|TIME_SECONDS
//--- 显示图表上和程序端日志中的消息
   Print(report);
   Comment(report);
//---
  }
//+------------------------------------------------------------------+
//| TesterDeinit函数                                                 |
//+------------------------------------------------------------------+
void OnTesterDeinit()
  {
//--- 优化持续时间
   string log_message=StringFormat("%s: optimization took %d seconds",
                                   __FUNCTION__,TimeLocal()-optimization_start);
```

© 2000-2025, MetaQuotes Ltd.

```
   PrintFormat(log_message);
   report=report+"\r\n"+log_message;
   Comment(report);
  }
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 初始化全局变量
   last_atr=0;
   last_body=0;
//--- 设置正确的交易量
   double min_lot=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   trade_lot=lots>min_lot? lots:min_lot;
//--- 创建ATR指标句柄
   atr_handle=iATR(_Symbol,_Period,ATRperiod);
   if(atr_handle==INVALID_HANDLE)
     {
      PrintFormat("%s: failed to create iATR, error code %d",__FUNCTION__,GetLastError
      return(INIT_FAILED);
     }
//--- EA成功初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 交易信号
   static int signal=0; // +1表示买入信号，-1 表示卖出信号
//--- 检查和关闭 'holdbars'柱形图之前开仓的旧持仓
   ClosePositionsByBars(holdbars,slippage,EXPERT_MAGIC);
//--- 检查新柱形图
   if(isNewBar())
     {
      //--- 检查信号是否存在
      signal=CheckSignal();
     }
//--- 如果持有单边持仓，跳过信号-等候直至关闭
   if(signal!=0 && PositionsTotal()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
     {
      signal=0;
      return; // 退出NewTick事件处理程序，并在新柱形图出现之前不要进入市场
     }
//--- 对于锁仓账户，每个持仓都是单独持仓和平仓
   if(signal!=0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 买入信号
      if(signal>0)
        {
         PrintFormat("%s: Buy signal! Revers=%s",__FUNCTION__,string(revers));
         if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
      //--- 卖出信号
      if(signal<0)
        {
         PrintFormat("%s: Sell signal! Revers=%s",__FUNCTION__,string(revers));
         if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
     }
//--- OnTick函数结束
  }
//+------------------------------------------------------------------+
//| 检查新的交易信号                                                 |
//+------------------------------------------------------------------+
int CheckSignal()
  {
//--- 0 意味着没有信号
   int res=0;
//--- 获得倒数第二个完整柱形图的ATR值（柱形图索引是2）
   double atr_value[1];
   if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
     {
      last_atr=atr_value[0];
      //--- 获得最后关闭的柱形图的MqlRates类型数组数据
      MqlRates bar[1];
      if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
        {
         //--- 计算最后完整柱形图的柱体大小
         last_body=bar[0].close-bar[0].open;
         //--- 如果最后柱形图的柱体（索引为1）超过的之前ATR的值（柱形图索引为2），那么会收到一个交易信
         if(MathAbs(last_body)>kATR*last_atr)
            res=last_body>0?1:-1; // 向上蜡烛图的正值
        }
      else
         PrintFormat("%s: Failed to receive the last bar! Error",__FUNCTION__,GetLastE
     }
   else
      PrintFormat("%s: Failed to receive ATR indicator value! Error",__FUNCTION__,GetL
//--- 如果启用了反向交易模式
   res=revers?-res:res;  // 在必要时反转信号（返回-1而不是1，反之亦然）
//--- 返回一个交易信号值
   return (res);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|  当新柱形图出现时返回'true'                                      |
//+------------------------------------------------------------------+
bool isNewBar(const bool print_log=true)
  {
   static datetime bartime=0; //存储当前柱形图的开盘时间
//--- 获得零柱的开盘时间
   datetime currbar_time=iTime(_Symbol,_Period,0);
//--- 如果开盘时间更改，则新柱形图出现
   if(bartime!=currbar_time)
     {
      bartime=currbar_time;
      lastbar_timeopen=bartime;
      //--- 在日志中显示新柱形图开盘时间的数据
      if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION)||MQLInfoInteger(MQL_TESTER)))
        {
         //--- 显示新柱形图开盘时间的信息
         PrintFormat("%s: new bar on %s %s opened at %s",__FUNCTION__,_Symbol,
                     StringSubstr(EnumToString(_Period),7),
                     TimeToString(TimeCurrent(),TIME_SECONDS));
         //--- 获取关于最后报价的数据
         MqlTick last_tick;
         if(!SymbolInfoTick(Symbol(),last_tick))
            Print("SymbolInfoTick() failed, error = ",GetLastError());
         //--- 显示最后报价的时间，精确至毫秒
         PrintFormat("Last tick was at %s.%03d",
                     TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%1000
        }
      //--- 我们有一个新柱形图
      return (true);
     }
//--- 没有新柱形图
   return (false);
  }
//+------------------------------------------------------------------+
//| 以指定交易量和市场价买入                                         |
//+------------------------------------------------------------------+
bool Buy(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- 以市场价买入
   return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| 以指定交易量和市场价卖出                                         |
//+------------------------------------------------------------------+
bool Sell(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- 以市场价卖出
   return (MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber));
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 在柱形图中根据持有时间平仓                                       |
//+------------------------------------------------------------------+
void ClosePositionsByBars(int holdtimebars,ulong deviation=10,ulong  magicnumber=0)
  {
   int total=PositionsTotal(); // 持仓数量
//--- 重复持仓
   for(int i=total-1; i>=0; i--)
     {
      //--- 持仓参数
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
      int bars=iBarShift(_Symbol,PERIOD_CURRENT,position_open)+1;

      //--- 如果持仓的生命周期很长，则幻数和交易品种匹配
      if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
         int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
         double volume=PositionGetDouble(POSITION_VOLUME);
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         string str_type=StringSubstr(EnumToString(type),14);
         StringToLower(str_type); //纠正消息格式的小写文本案例
         PrintFormat("Close position #%I64u %s %s %.2f",
                     position_ticket,position_symbol,str_type,volume);
         //--- 设置一个订单类型并发送交易请求
         if(type==POSITION_TYPE_BUY)
            MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber,position_ticket);
         else
            MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber,position_ticket);
        }
     }
  }
//+------------------------------------------------------------------+
//| 准备和发送交易请求                                               |
//+------------------------------------------------------------------+
bool MarketOrder(ENUM_ORDER_TYPE type,double volume,ulong slip,ulong magicnumber,ulong
  {
//--- 声明和初始化结构
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(type==ORDER_TYPE_BUY)
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- 请求参数
   request.action   =TRADE_ACTION_DEAL;                        // 交易操作类型
   request.position =pos_ticket;                               // 关闭情况下的持仓单号
```

© 2000-2025, MetaQuotes Ltd.

```
   request.symbol    =Symbol();                                    // 交易品种
   request.volume    =volume;                                      // 交易量
   request.type      =type;                                        // 订单类型
   request.price     =price;                                       // 交易价格
   request.deviation=slip;                                         // 可允许的价格偏差
   request.magic     =magicnumber;                                 // 订单幻数
//--- 发送请求
   if(!OrderSend(request,result))
     {
      //--- 显示数据失败
      PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
                  request.symbol,EnumToString(type),volume,request.price,GetLastError
      return (false);
     }
//--- 通知成功操作
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
   return (true);
  }
```

另见

测试交易策略、处理优化结果、OnTesterDeinit、OnTesterPass、ParameterGetRange、ParameterSetRange

© 2000-2025, MetaQuotes Ltd.

# OnTesterDeinit

在EA优化之后，当TesterDeinit事件发生时在EA中调用这个函数。

```
void  OnTesterDeinit(void);
```

返回值

无返回值

注意

TesterDeinit事件在策略测试中的EA优化完成后生成。

带有OnTesterDeInit()或OnTesterPass()事件处理程序的EA会在优化开始期间，在单独的程序端图表上自动下载。它包括测试中已经制定的交易品种和周期。该函数的设计目的是为最终处理所有的优化结果。

请记住由测试代理使用FrameAdd()函数发送的优化框架可能会绑定在一起，并需要时间来交付。因此，不是所有的框架，以及TesterPass事件，可能在优化结束之前到达，并在OnTesterPass()中处理的。如果您想要得到OnTesterDeinit()中的所有迟到的框架，请使用FrameNext()函数来放置代码块。

另见

测试交易策略、处理优化结果、TesterStatistics、OnTesterInit、OnTesterPass、ParameterGetRange、ParameterSetRange

© 2000-2025, MetaQuotes Ltd.

# OnTesterPass

当 TesterPass事件发生时，在EA中调用这个函数来处理EA优化期间的新数据框架。

```
void  OnTesterPass(void);
```

返回值

无返回值

注意

TesterPass事件在接收策略测试中EA优化期间的框架时自动生成。

带有OnTesterDeInit()或OnTesterPass()事件处理程序的EA会在优化开始期间，在单独的程序端图表上自动下载。它包括测试中已经制定的交易品种和周期。这个函数意味着处理优化期间从测试代理接收的框架。包含测试结果的框架应该使用FrameAdd()函数，从OnTester()处理程序来发送。

请记住由测试代理使用FrameAdd()函数发送的优化框架可能会绑定在一起，并需要时间来交付。因此，不是所有的框架，以及TesterPass事件，可能在优化结束之前到达，并在OnTesterPass()中处理的。如果您想要得到OnTesterDeinit()中的所有迟到的框架，请使用FrameNext()函数来放置代码块。

完成OnTesterDeinit()优化之后，可以使用FrameFirst()/FrameFilter和FrameNext()函数再次分类所有接收的框架。

另见

测试交易策略、处理优化结果、OnTesterInit、OnTesterDeinit、FrameFirst、FrameFilter、FrameNext、FrameInputs

© 2000-2025, MetaQuotes Ltd.

# 获取市场信息

函数用来接收关于市场规则的信息。

| 函数 | 功能 |
| --- | --- |
| SymbolsTotal | 返回有效交易品种数量（选择市场报价或者全部） |
| SymbolExist | 检查指定名称的交易品种是否存在 |
| SymbolName | 返回指定交易品种名称 |
| SymbolSelect | 市场报价窗口选择或者去除交易品种 |
| SymbolIsSynchronized | 检测程序端选择的交易品种的数据与交易服务器的数据是否同步。 |
| SymbolInfoDouble | 为相关属性返回交易品种双精度值 |
| SymbolInfoInteger | 为相关属性返回指定交易品种的整数型值（长整型，日期时间，整型或者布尔型） |
| SymbolInfoString | 为相关属性返回指定交易品种的字符串类型值 |
| SymbolInfoMarginRate | 根据订单类型和方向返回预付款比率 |
| SymbolInfoTick | 返回MqlTick型变量中指定交易品种当前价格。 |
| SymbolInfoSessionQuote | 平日允许为指定交易品种接收指定报价期的起止时间 |
| SymbolInfoSessionTrade | 平日允许为指定交易品种接收指定交易期的起止时间 |
| MarketBookAdd | 提供选定交易品种市场深度开盘，签署DOM更改的接收通告 |
| MarketBookRelease | 提供选定交易品种市场深度收盘，取消DOM更改的接收通告的签署。 |
| MarketBookGet | 返回包括指定交易品种市场深度记录的结构数组MqlBookInfo |

© 2000-2025, MetaQuotes Ltd.

# 交易品种总计

返回可用交易品种数量（在市场调查中选择）。

```
int  SymbolsTotal(
   bool  selected      // True - 只是市场报价中的交易品种
   );
```

参量

*selected*

[in] 请求模式，可以是true或false。

返回值

如果“选择”参量是true,函数返回字符的数量就在市场窗口中选择，如果值是false，返回所有交易品种的数量。

示例：

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获取可用交易品种数量
   int total    = SymbolsTotal(false);       // 服务器上的所有交易品种
   int selected = SymbolsTotal(true);        // 添加到市场报价窗口的所有交易品种

//--- 将获得的数据发送到日志
   PrintFormat("Number of available symbols on the server: %d\n"+
               "Number of available symbols selected in MarketWatch: %d", total, selec
   /*
   result:
   Number of available symbols on the server: 140
   Number of available symbols selected in MarketWatch: 11
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolExist

检查指定名称的交易品种是否存在。

```
bool  SymbolExist(
   const string  name,     // 交易品种名称
   bool&   is_custom       // 自定义交易品种属性
   );
```

参数

*名称*

[in] 交易品种名称。

*is_custom*

[out] 成功执行时设置自定义交易品种属性。如果为true，检测的交易品种就是自定义交易品种。

返回值

如果为false，则意味着在标准和自定义交易品种中没有找到该交易品种。

示例：

```
#define SYMBOL_NAME "GBPUSDn"
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明自定义交易品种标识，并检查是否存在其名称在SYMBOL_NAME中指定的交易品种
   bool custom = false;
   bool result = SymbolExist(SYMBOL_NAME, custom);

//--- 声明默认的 "找不到交易品种"消息文本
   string text = StringFormat("The symbol '%s' was not found among either the standard

//--- 如果找到交易品种，则根据找到交易品种的列表创建消息文本
   if(result)
     {
      //--- 如果这是一个标准交易品种
      if(!custom)
         text = StringFormat("The '%s' symbol is available on the server.", SYMBOL_NAM
      //--- 如果这是一个自定义交易品种
      else
         text = StringFormat("The symbol '%s' was found in the list of custom symbols.
     }

//--- 将有关检查结果的消息发送到日志
   Print(text);
   /*
```

© 2000-2025, MetaQuotes Ltd.

```
   result for standard 'GBPUSD' symbol:
   The 'GBPUSD' symbol is available on the server.

   result for custom 'GBPUSDx' symbol:
   The symbol 'GBPUSDx' was found in the list of custom symbols.

   result for missing 'GBPUSDn' symbol:
   The symbol 'GBPUSDn' was not found among either the standard or custom symbols.
   */
  }
```

另见

SymbolsTotal, SymbolSelect, Custom symbols

© 2000-2025, MetaQuotes Ltd.

# SymbolName

返回交易品种名称。

```
string  SymbolName(
   int    pos,           // 列表中的数字
   bool   selected       // true - 只是市场报价中的交易品种
   );
```

参量

*pos*

[in] 一个教育品种的订单编号。

*selected*

[in] 请求方式，如果值是true,该交易品种从其在市场观测的选择中找出，如果值是false，交易品种从总目录中找出。

返回值

包含交易品种名称的字符串的值。

示例：

```
#define SYMBOL_NAME "GBPHKD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置服务器交易品种搜索结果的标识
   bool found = false;

//--- 在所有服务器交易品种列表中找出 "SYMBOL_NAME"交易品种
   int total = SymbolsTotal(false);
   for(int i=0; i<total; i++)
     {
      //--- 通过循环索引获取列表中交易品种的名称
      string name = SymbolName(i, false);

      //--- 如果这是所需的交易品种，则将其名称和在列表中的位置发送到日志并退出循环
      if(name == SYMBOL_NAME)
        {
         PrintFormat("The '%s' symbol was found in the list of server symbols at index
         found = true;
         break;
        }
     }

//--- 如果在服务器上找不到交易品种，请在关闭之前进行报告
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!found)
      PrintFormat("The '%s' symbol was not found on the server.", SYMBOL_NAME);
}
```

© 2000-2025, MetaQuotes Ltd.

# SymbolSelect

在市场观测中选择一个交易品种，或者从窗口中移动一个交易品种。

```
bool  SymbolSelect(
   string  name,        // 交易品种名称
   bool    select       // 添加或者移除
   );
```

参量

*名称*

[in] 交易品种名称。

*select*

[in]转换，如果值是false，交易品种从市场观测中删除，然后在窗口中选择一个交易品种，如果交易品种共图表是开放式的，交易品种就不能被删除，或者该交易品种有开放定位。

返回值

以防返回失败。

示例：

```
#define SYMBOL_NAME "GBPHKD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 检查列表中交易品种是否存在，如果没有找到，进行报告并完成工作
   bool custom = false;
   if(!SymbolExist(SYMBOL_NAME, custom))
     {
      PrintFormat("'%s' symbol not found in the lists", SYMBOL_NAME);
      return;
     }

//--- 将交易品种添加到市场报价窗口
   ResetLastError();
   if(!SymbolSelect(SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }
//--- 如果交易品种被成功地添加到列表中，请在市场报价窗口中获取它的索引号，并将结果发送到日志
   int index = SymbolIndex(SYMBOL_NAME);
   PrintFormat("The '%s' symbol has been added to the MarketWatch list. Symbol index i

//--- 现在从市场报价窗口移除交易品种
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
   if(!SymbolSelect(SYMBOL_NAME, false))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }
//--- 如果交易品种被成功地从列表中移除，则市场报价窗口的索引号-1，并将删除结果发送到日志
   index = SymbolIndex(SYMBOL_NAME);
   PrintFormat("The '%s' symbol has been removed from the MarketWatch list. Symbol inc

   /*
   result:
   The 'GBPHKD' symbol has been added to the MarketWatch list. Symbol index in the lis
   The 'GBPHKD' symbol has been removed from the MarketWatch list. Symbol index in the
   */
  }
//+------------------------------------------------------------------+
//| 返回交易品种在市场报价交易品种列表中的索引号                     |
//+------------------------------------------------------------------+
int SymbolIndex(const string symbol)
  {
   int total = SymbolsTotal(true);
   for(int i=0; i<total; i++)
     {
      string name = SymbolName(i, true);
      if(name == symbol)
         return i;
     }
   return(WRONG_VALUE);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 同步交易品种

该函数检验在客户端中选出来的交易品种数据与交易服务器中的数据是同步的。

```
bool  SymbolIsSynchronized(
   string  name,        // 交易品种名称
   );
```

参量

*名称*

[in] 交易品种名称。

返回值

如果数据是 同步的，返回true,否则返回false。

示例：

```
#define SYMBOL_NAME "EURUSD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 使用“SYMBOL_NAME”交易品种获取用于同步程序端数据和服务器数据的标识
   bool    sync = SymbolIsSynchronized(SYMBOL_NAME);

//--- 根据同步标识创建消息
   string text = StringFormat("The data on the '%s' symbol in the terminal is synchron
   if(!sync)
      text = StringFormat("The data for the '%s' symbol in the terminal is not synchr

//--- 将获得的结果发送到日志
   Print(text);

   /*
   result for synchronized data:
   The data on the 'EURUSD' symbol in the terminal is synchronized with the data on th

   result for data not yet synchronized:
   The data for the 'GBPHKD' symbol in the terminal is not synchronized with the data
   */
  }
```

另见

SymbolInfoInterger, 组织数据接入

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoDouble

返回一个规定交易样品的类似特性，函数有两个变量。

1. 快速返回规定值。

```
double  SymbolInfoDouble(
   string                    name,         // 交易品种
   ENUM_SYMBOL_INFO_DOUBLE   prop_id       // 属性标识符
   );
```

2. 返回真值或失败值取决于函数操作是否成功，如果成功，属性值通过引用从最后的参量传递到接受变量中。

```
bool  SymbolInfoDouble(
   string                    name,         // 交易品种
   ENUM_SYMBOL_INFO_DOUBLE   prop_id,      // 属性标识符
   double&                   double_var    // 这里假设属性值
   );
```

参量

*name*

[in] 交易品种名称

*prop_id*

[in] 交易样品属性标识符，值可能是计算式 ENUM_SYMBOL_INFO_DOUBLE 中的一个。

*double_var*

[out]双精度型变量接收需求属性的值

返回值

双精度型值。如果执行失败，使用GetLastError()函数可以获得有关的错误信息：

- 5040 -指定交易品种名称的无效字符串参数，
- 4301 -未知交易品种（金融工具），
- 4302 -交易品种未选进“市场报价”（可用列表中未发现），
- 4303 -交易品种属性的无效标识符。

注意

如果函数用于获得最近订单号信息，推荐使用SymbolInfoTick()函数。由于程序端连接交易账户，那么很可能不会出现单引号。这种情况下，请求的值将不会确定。

在大部分情况下，使用 SymbolInfoTick() 函数足以允许用户接收买价，买价，收盘价，交易量的值和单次调用时最近订单号的到达时间。

SymbolInfoMarginRate()函数根据订单类型和方向提供有关所收取预付款金额的数据。

示例：

```
void OnTick()
  {
//--- 从交易品种属性获得点差
```

© 2000-2025, MetaQuotes Ltd.

```
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- 现在自己计算点差
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoInteger

返回一个规定交易样品的类似特质，该函数有两种变体

1. 快速返回属性值

```
long  SymbolInfoInteger(
   string                     name,       // 交易品种
   ENUM_SYMBOL_INFO_INTEGER   prop_id     // 属性标识符

   );
```

2. 返回真值或失败值取决于函数是否成功执行，如果成功，属性值通过引用从最后的参量传递到接受变量中

```
bool  SymbolInfoInteger(
   string                     name,       // 交易品种
   ENUM_SYMBOL_INFO_INTEGER   prop_id,    // 属性标识符
   long&                      long_var    // 这里假设属性值
   );
```

参量

*name*

[in] 交易样品名称

*prop_id*

[in] 交易样品属性标识符，值可以是计算式中 ENUM_SYMBOL_INFO_INTEGER 的一个。

*long_var*

[out] 长型变量接收要求变量的值。

返回值

长整型值 如果执行失败，使用GetLastError()函数可以获得有关的 错误 信息：

- 5040 -指定交易品种名称的无效字符串参数，
- 4301 -未知交易品种 (金融工具)，
- 4302 -交易品种未选进 "市场报价" (可用列表中未发现)，
- 4303 -交易品种属性的无效标识符。

注意

如果函数用于获得最近订单号信息，推荐使用 SymbolInfoTick() 函数。由于程序端连接交易账户，那么很可能不会出现单引号。这种情况下，请求的值将不会确定。

在大部分情况下，使用 SymbolInfoTick() 函数足以允许用户接收买价，买价，收盘价，交易量的值和单次调用时最近订单号的到达时间。

示例：

```
void OnTick()
  {
//--- 从交易品种属性获得点差
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
```

© 2000-2025, MetaQuotes Ltd.

```
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- 现在自己计算点差
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoString

一个规定交易品种返回相对应的特性，该函数中有两个变体。

1. 立即返回属性值

```
string  SymbolInfoString(
   string                     name,        // 交易品种
   ENUM_SYMBOL_INFO_STRING    prop_id      // 属性标识符
   );
```

2. 返回true或者false，由该函数运行成功与否来决定，如果成功，在培育变量通过引用传递到最后常量中函数属性固定。

```
bool  SymbolInfoString(
   string                     name,        // 交易品种
   ENUM_SYMBOL_INFO_STRING    prop_id,     // 属性标识符
   string&                    string_var   // 这里假设属性值
   );
```

参量

*名称*

[in] 交易品种名称。

*prop_id*

[in] 一个交易品种属性标识符，值是 ENUM_SYMBOL_INFO_STRING 其中一个。

*string_var*

[out] 字符串类型变量接收需求属性的值。

返回值

字符串的值。如果执行失败，使用GetLastError()函数可以获得有关的错误信息：

- 5040 -指定交易品种名称的无效字符串参数，
- 4301 -未知交易品种（金融工具），
- 4302 -交易品种未选进“市场报价”（可用列表中未发现），
- 4303 -交易品种属性的无效标识符。

注意

如果函数用于获得最近订单号信息，推荐使用SymbolInfoTick()函数。由于程序端连接交易账户，那么很可能不会出现单引号。这种情况下，请求的值将不会确定。

在大部分情况下，使用 SymbolInfoTick() 函数足以允许用户接收买价，买价，收盘价，交易量的值和单次调用时最近订单号的到达时间。

示例：

```
#define SYMBOL_NAME "USDJPY"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
//--- 获取SYMBOL_NAME交易品种的字符串数据
   string currency_base   = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_BASE);     //
   string currency_profit = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_PROFIT);   //
   string currency_margin = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_MARGIN);   //
   string symbol_descript = SymbolInfoString(SYMBOL_NAME, SYMBOL_DESCRIPTION);       //

//--- 使用获得的数据创建消息文本
   string text=StringFormat("Symbol %s:\n"+
                            "- Currency Base: %s\n"+
                            "- Currensy Profit: %s\n"+
                            "- Currency Margin: %s\n"+
                            "- Symbol Description: %s",
                            SYMBOL_NAME, currency_base,
                            currency_profit, currency_margin,
                            symbol_descript);

//--- 将获得的数据发送到日志
   Print(text);
   /*
   Symbol USDJPY:
   - Currency Base: USD
   - Currensy Profit: JPY
   - Currency Margin: USD
   - Symbol Description: US Dollar vs Yen
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoMarginRate

根据订单类型和方向返回预付款比率。

```
bool  SymbolInfoMarginRate(
   string              name,                    // 交易品种名称
   ENUM_ORDER_TYPE     order_type,              // 订单类型
   double&             initial_margin_rate,     // 初始预付款比率
   double&             maintenance_margin_rate  // 维持预付款比率
   );
```

参数

*名称*

[in] 交易品种名称。

*order_type*

[in] 订单类型。

*initial_margin_rate*

[in] 用于接收初始预付款比率的双精度类型变量。初始预付款是相应方向的证券一手交易的保证金。初始预付款与比率相乘后，我们会收到下单指定类型订单时账户上保留的资金量。.

*maintenance_margin_rate*

[out] 用于接收维持预付款比率的双精度类型变量。维持预付款是维护相应方向一手持仓的最低数额。维持预付款与比率相乘后，我们会收到激活指定类型订单后，账户上保留的资金量。

返回值

如果属性请求成功，返回true，否则false。

示例：

```
#define SYMBOL_NAME Symbol()

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明设置比率的变量
   double initial_margin_rate = 0;      // 初始预付款比率
   double maintenance_margin_rate = 0; // 维持预付款比率

//--- 获取并发送买入市价单的SYMBOL_NAME交易品种比率到日志
   SymbolInfoMarginRatePrint(SYMBOL_NAME, ORDER_TYPE_BUY, initial_margin_rate, mainter

//--- 获取并发送卖出市价单的SYMBOL_NAME交易品种比率到日志
   SymbolInfoMarginRatePrint(SYMBOL_NAME, ORDER_TYPE_SELL, initial_margin_rate, mainte
```

© 2000-2025, MetaQuotes Ltd.

```
   /*
   result:
   Initial margin rate for symbol AFKS for the Buy market order: 0.225600
   Maintenance margin rate for symbol AFKS for the Buy market order: 0.112800
   Initial margin rate for symbol AFKS for the Sell market order: 0.254400
   Maintenance margin rate for symbol AFKS for the Sell market order: 0.127200
   */
  }
//+------------------------------------------------------------------+
//| 将预付款比率发送到日志                                           |
//+------------------------------------------------------------------+
void SymbolInfoMarginRatePrint(const string symbol, const ENUM_ORDER_TYPE order_type,
  {
//--- 获取order_type市价单的'symbol'比率
   ResetLastError();
   if(!SymbolInfoMarginRate(symbol, order_type, initial_margin_rate, maintenance_margi
     {
      Print("SymbolInfoMarginRate() failed. Error ", GetLastError());
      return;
     }
//--- 记录获得的比率值
   string type=(order_type==ORDER_TYPE_BUY ? "Buy" : order_type==ORDER_TYPE_SELL ? "Se
   PrintFormat("Initial margin rate for symbol %s for the %s market order: %f\n"+
               "Maintenance margin rate for symbol %s for the %s market order: %f",
               SYMBOL_NAME, type, initial_margin_rate,
               SYMBOL_NAME, type, maintenance_margin_rate);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoTick

该函数返回在MqlTick类型常量中的规定交易品种的当前价值。

```
bool  SymbolInfoTick(
   string    symbol,      // 交易品种名称
   MqlTick&  tick         // 结构参考
   );
```

参量

*交易品种*

[in] 交易品种名称。

*tick*

[out] 连接 MqlTick 类型结构，显示当前价格和最后价格的即时更新。

返回值

如果成功，函数返回真值，否则返回失败

示例：

```
#define SYMBOL_NAME "EURUSD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明一个维度为1的MqlTick结构类型的数组
   MqlTick tick[1]={};

//--- 将SYMBOL_NAME交易品种的最新价格获取到MqlTick结构中
   if(!SymbolInfoTick(SYMBOL_NAME, tick[0]))
     {
      Print("SymbolInfoTick() failed. Error ", GetLastError());
      return;
     }

//--- 将获得的数据发送到日志
   PrintFormat("Latest price data for the '%s' symbol:", SYMBOL_NAME);
   ArrayPrint(tick);
   /*
   result:
   Latest price data for the 'EURUSD' symbol:
                    [time]   [bid]   [ask] [last] [volume]    [time_msc] [flags] [volu
   [0] 2024.05.17 23:58:54 1.08685 1.08695 0.0000        0 1715990334319       6
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoSessionQuote

允许为规定交易样品和周期接收引用登陆的开始和结束时间。

```
bool  SymbolInfoSessionQuote(
   string            name,                // 交易品种名称
   ENUM_DAY_OF_WEEK  day_of_week,         // 一周中的每天
   uint              session_index,       // 期指
   datetime&         from,                // 期始时间
   datetime&         to                   // 期末时间
   );
```

参量

*名称*

[in] 交易品种名称

*ENUM_DAY_OF_WEEK*

[in]每周日期，计算式 ENUM_DAY_OF_WEEK值、

*无符号整型(uint)*

[in] 登陆的序列号，接收开始和结束的时间，以0开始登陆

*从*

[out] 从00小时00分钟以短时间开始登陆，返回值的日期可以忽略不计

*到*

[out] 从00小时00分钟以短时间结束登陆，返回值的日期可以忽略不计

返回值

如果接收到规定登陆数据的交易品种和每周日期，返回真值，否则返回false

示例：

```
#define SYMBOL_NAME    Symbol()
#define SESSION_INDEX 0

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 记录带有交易品种和SESSION_INDEX的标题
//--- 从周一到周五循环，在日志中记录报价时段的开始和结束时间
   PrintFormat("Symbol %s, Quote session %d:", SYMBOL_NAME, SESSION_INDEX);
   for(int i=MONDAY; i<SATURDAY; i++)
      SymbolInfoSessionQuotePrint(SYMBOL_NAME, (ENUM_DAY_OF_WEEK)i, SESSION_INDEX);
   /*
   result:
```

© 2000-2025, MetaQuotes Ltd.

```
   Symbol RU000A103661, Quote session 0:
   - Monday     06:45 - 00:00
   - Tuesday    06:45 - 00:00
   - Wednesday  06:45 - 00:00
   - Thursday   06:45 - 00:00
   - Friday     06:45 - 00:00
   */
  }
//+------------------------------------------------------------------+
//| 发送指定报价时段的开始和结束时间                                 |
//| 到日志 (指定交易品种，工作日)                                    |
//+------------------------------------------------------------------+
void SymbolInfoSessionQuotePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- 声明变量来记录报价时段的开始和结束
   datetime date_from;  // 时段开始时间
   datetime date_to;    // 时段结束时间

//--- 从报价时段按交易品种和星期几获取数据
   if(!SymbolInfoSessionQuote(symbol, day_of_week, session_index, date_from, date_to))
     {
      Print("SymbolInfoSessionQuote() failed. Error ", GetLastError());
      return;
     }

//--- 从枚举常量创建星期名称
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- 将指定报价时段的数据发送到日志
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Tir
  }
```

另见

交易品种属性，TimeToStruct，数据结构

© 2000-2025, MetaQuotes Ltd.

# SymbolInfoSessionTrade

平日指定交易品种允许接收规定交易的起始和结束时间

```
bool  SymbolInfoSessionTrade(
   string            name,                // 交易品种名称
   ENUM_DAY_OF_WEEK  day_of_week,         // 一周中的每天
   uint              session_index,       // 期指
   datetime&         from,                // 期始时间
   datetime&         to                   // 期末时间
   );
```

参量

*名称*

[in] 交易品种名称。

*ENUM_DAY_OF_WEEK*

[in] 每周日期，计算 ENUM_DAY_OF_WEEK值

*无符号整型(uint)*

[in] 一段登陆的数字，能够收到起始和结束时间，从0开始

*从*

[out] 从00小时00分钟以短时间开始登陆，返回值的日期可以忽略不计。

*到*

[out] 从00小时00分钟以短时间结束登陆，返回值的日期可以忽略不计。

返回值

如果接收到规定登陆数据的交易品种和每周日期，返回真值，否则返回false。

示例：

```
#define SYMBOL_NAME    Symbol()
#define SESSION_INDEX 0

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 记录带有交易品种和SESSION_INDEX的标题
//--- 从周一到周五循环，在日志中记录交易时段的开始和结束时间
   PrintFormat("Symbol %s, Trade session %d:", SYMBOL_NAME, SESSION_INDEX);
   for(int i=MONDAY; i<SATURDAY; i++)
      SymbolInfoSessionTradePrint(SYMBOL_NAME, (ENUM_DAY_OF_WEEK)i, SESSION_INDEX);
   /*
   result:
   Symbol GBPUSD, Trade session 0:
   - Monday     00:15 - 23:55
```

© 2000-2025, MetaQuotes Ltd.

```
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   */
  }
//+------------------------------------------------------------------+
//| 发送指定交易时段的开始和结束时间                                 |
//| 到日志 (指定交易品种，工作日)                                    |
//+------------------------------------------------------------------+
void SymbolInfoSessionTradePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- 声明变量来记录交易时段的开始和结束
   datetime date_from;  // 时段开始时间
   datetime date_to;    // 时段结束时间

//--- 从交易时段按交易品种和星期几获取数据
   if(!SymbolInfoSessionTrade(symbol, day_of_week, session_index, date_from, date_to))
     {
      Print("SymbolInfoSessionTrade() failed. Error ", GetLastError());
      return;
     }

//--- 从枚举常量创建星期名称
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- 将指定交易时段的数据发送到日志
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Tir
  }
```

另见

交易品种属性，TimeToStruct，数据结构

© 2000-2025, MetaQuotes Ltd.

# MarketBookAdd

提供所选的交易品种的开盘市场深度信息，预定接收DOM转变的通知。

```
bool  MarketBookAdd(
   string  symbol      // 交易品种
   );
```

参量

*交易品种*

[in]交易品种的名称，市场深度用于EA交易或脚本中。

返回值

如果成功开仓，返回true，否则是false。

整数

一般来说，该函数一定从 OnInit() 函数中调用或者在分类构造函数中。为处理引用警报，在EA交易程序中必须包括 OnBookEvent函数(字符串& 交易品种)。

示例：

```
#define   SYMBOL_NAME   "GBPUSD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打开SYMBOL_NAME交易品种的市场深度
   if(!MarketBookAdd(SYMBOL_NAME))
     {
      PrintFormat("MarketBookAdd(%s) failed. Error ", SYMBOL_NAME, GetLastError());
      return;
     }

//--- 将成功打开市场深度的消息发送到日志
   PrintFormat("The MarketBook for the '%s' symbol was successfully opened and a subsc

//--- 等待2秒
   Sleep(2000);

//--- 完成后，取消订阅打开的市场深度
   ResetLastError();
   if(MarketBookRelease(SYMBOL_NAME))
      PrintFormat("MarketBook for the '%s' symbol was successfully closed", SYMBOL_NAM
   else
      PrintFormat("Error %d occurred when closing MarketBook using the '%s' symbol", G

   /*
```

© 2000-2025, MetaQuotes Ltd.

```
   result:
   The MarketBook for the 'GBPUSD' symbol was successfully opened and a subscription t
   MarketBook for the 'GBPUSD' symbol was successfully closed
   */
  }
```

另见

市场深度结构，结构和类

© 2000-2025, MetaQuotes Ltd.

# MarketBookRelease

提供所选交易品种的收盘市场报价信息，并取消预定接收DOM转变的通知。

```
bool  MarketBookRelease(
   string  symbol      // 交易品种
   );
```

参量

*交易品种*

[in]交易品种名称。

返回值

如果成功关闭是真值，否则是错误值。

注意

通常，该函数需要调用OnDeinit()函数才能启用，如果在OnInit()函数中调用类似于MarketBookAdd()函数，就会调用分类解构函数，从分类解构函数中调用类似于MarketBookAdd()的函数。

示例：

```
#define SYMBOL_NAME "GBPUSD"

//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打开SYMBOL_NAME交易品种的市场深度
   if(!MarketBookAdd(SYMBOL_NAME))
     {
      PrintFormat("MarketBookAdd(%s) failed. Error ", SYMBOL_NAME, GetLastError());
      return;
     }

//--- 将成功打开市场深度的消息发送到日志
   PrintFormat("The MarketBook for the '%s' symbol was successfully opened and a subsc

//--- 等待2秒
   Sleep(2000);

//--- 完成后，取消订阅打开的市场深度
//--- 将有关成功取消订阅市场深度或有关错误的消息发送到日志
   ResetLastError();
   if(MarketBookRelease(SYMBOL_NAME))
      PrintFormat("MarketBook for the '%s' symbol was successfully closed", SYMBOL_NAM
   else
      PrintFormat("Error %d occurred when closing MarketBook using the '%s' symbol", G
```

© 2000-2025, MetaQuotes Ltd.

```
   /*
   result:
   The MarketBook for the 'GBPUSD' symbol was successfully opened and a subscription t
   MarketBook for the 'GBPUSD' symbol was successfully closed
   */
  }
```

另见

市场深度结构，结构和类

© 2000-2025, MetaQuotes Ltd.

# MarketBookGet

返回结构数组 MqlBookInfo 包括指定交易品种的市场报价记录。

```
bool  MarketBookGet(
   string         symbol,      // 交易品种
   MqlBookInfo&   book[]       // 参考数组
   );
```

参量

*symbol*

[in] 交易品种名称。

*book[]*

[in] 关于市场报价的记录。数组再分配给记录数量。如果 动态数组 没有再分配给操作内存，客户端会自行分配数组。

返回值

如果成功，返回true，否则是false。

整数

市场深度通过 MarketBookAdd() 函数重启。

示例：

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo for ",Symbol());
      for(int i=0;i<size;i++)
        {
         Print(i+":",priceArray[i].price
               +"    Volume = "+priceArray[i].volume,
               " type = ",priceArray[i].type);
        }
     }
   else
     {
      Print("Could not get contents of the symbol DOM ",Symbol());
     }
```

另见

市场深度结构，结构和类

© 2000-2025, MetaQuotes Ltd.

# 经济日历函数

这部分描述了处理可在MetaTrader平台直接使用的经济日历的函数。经济日历是一个即时可用的百科全书，描述了宏观经济指标、指标发布日期以及重要程度。宏观经济指标的相关值会在发布时立即发送至MetaTrader平台，并以标签的形式显示在图表上，使您可以根据国家、货币和重要性直观地跟踪所需的指标。

所有用于经济日历的函数都使用交易服务器时间(TimeTradeServer)。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

经济日历函数可以根据自定义重要性标准，从必要的国家/货币的角度对传入事件进行自动分析。

| 函数 | 操作 |
| --- | --- |
| CalendarCountryById | 根据ID获得国家描述 |
| CalendarEventById | 根据ID获得事件描述 |
| CalendarValueById | 根据ID获得事件值描述 |
| CalendarCountries | 获得日历中可用的国家名称数组 |
| CalendarEventByCountry | 根据指定国家代码获得日历中可用的所有事件描述数组 |
| CalendarEventByCurrency | 根据指定货币获得日历中可用的所有事件描述数组 |
| CalendarValueHistoryByEvent | 根据事件ID获得指定时期的所有事件值数组 |
| CalendarValueHistory | 通过根据国家和/或货币进行排序的能力，获得指定时期的所有事件值数组 |
| CalendarValueLastByEvent | 通过指定change_id，从日历数据库状态根据ID获得事件值数组 |
| CalendarValueLast | 通过指定change_id和根据国家和/或货币进行排序的能力，从日历数据库状态获得所有事件值数组 |

© 2000-2025, MetaQuotes Ltd.

# CalendarCountryById

根据ID获得国家描述。

```
bool  CalendarCountryById(
   const long           country_id,     //国家 ID
   MqlCalendarCountry&  country         // 用于接收国家描述的变量
   );
```

参数

*country_id*

[in] 国家ID (ISO 3166-1)。

*country*

[out] MqlCalendarCountry类型变量，用于接收国家描述。

返回值

如果成功返回true，否则返回false。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 5402 -ERR_CALENDAR_NO_DATA (国家未找到)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 从经济日历中获得国家列表
   MqlCalendarCountry countries[];
   int count=CalendarCountries(countries);
//--- 检查结果
   if(count==0)
      PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
//--- 如果有两个或两个以上的国家
   if(count>=2)
     {
      MqlCalendarCountry country;
      //--- 现在根据ID获得国家描述
      if(CalendarCountryById(countries[1].id, country))
        {
         //--- 准备国家描述
         string descr="id = "+IntegerToString(country.id)+"\n";
         descr+=("name = " + country.name+"\n");
         descr+=("code = " + country.code+"\n");
         descr+=("currency = " + country.currency+"\n");
         descr+=("currency_symbol = " + country.currency_symbol+"\n");
```

© 2000-2025, MetaQuotes Ltd.

```
         descr+=("url_name = " + country.url_name);
         //--- 显示国家描述
         Print(descr);
        }
      else
         Print("CalendarCountryById() failed. Error ",GetLastError());
     }
//---
  }
/*
  结果:
   id = 999
   name = European Union
   code = EU
   currency = EUR
   currency_symbol = €
   url_name = european-union
*/
```

另 见

CalendarCountries, CalendarEventByCountry

© 2000-2025, MetaQuotes Ltd.

# CalendarEventById

根据ID获得事件描述。

```
bool  CalendarEventById(
   ulong                event_id,      // 事件ID
   MqlCalendarEvent&    event          // 用于接收事件描述的变量
   );
```

参数

*event_id*

[in] 事件ID。

*event*

[out] MqlCalendarEvent类型变量，用于接收事件描述。

返回值

如果成功返回true，否则返回false。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR（一般运行时错误）。
- 5402 -ERR_CALENDAR_NO_DATA（国家未找到）。
- 5401 -ERR_CALENDAR_TIMEOUT（超过请求时限）。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 德国国家代码(ISO 3166-1 Alpha-2)
   string germany_code="DE";
//--- 获得德国事件
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(germany_code,events);
//--- 在“日志”中显示德国事件
   if(events_count>0)
     {
      PrintFormat("Germany events: %d",events_count);
      ArrayPrint(events);
     }
   else
     {
      PrintFormat("Failed to receive events for the country code %s, error %d",
                  germany_code,GetLastError());
      //--- 脚本提前完成
      return;
     }
//--- 从events[]数组获得最后事件的描述
```

© 2000-2025, MetaQuotes Ltd.

```
   MqlCalendarEvent event;
   ulong event_id=events[events_count-1].id;
   if(CalendarEventById(event_id,event))
     {
      MqlCalendarCountry country;
      CalendarCountryById(event.country_id,country);
      PrintFormat("Event description with event_id=%d received",event_id);
      PrintFormat("Country: %s (country code = %d)",country.name,event.country_id);
      PrintFormat("Event name: %s",event.name);
      PrintFormat("Event code: %s",event.event_code);
      PrintFormat("Event importance: %s",EnumToString((ENUM_CALENDAR_EVENT_IMPORTANCE)
      PrintFormat("Event type: %s",EnumToString((ENUM_CALENDAR_EVENT_TYPE)event.type))
      PrintFormat("Event sector: %s",EnumToString((ENUM_CALENDAR_EVENT_SECTOR)event.se
      PrintFormat("Event frequency: %s",EnumToString((ENUM_CALENDAR_EVENT_FREQUENCY)ev
      PrintFormat("Event release mode: %s",EnumToString((ENUM_CALENDAR_EVENT_TIMEMODE)
      PrintFormat("Event measurement unit: %s",EnumToString((ENUM_CALENDAR_EVENT_UNIT)
      PrintFormat("Number of decimal places: %d",event.digits);
      PrintFormat("Event multiplier: %s",EnumToString((ENUM_CALENDAR_EVENT_MULTIPLIER)
      PrintFormat("Source URL: %s",event.source_url);
     }
   else
      PrintFormat("Failed to get event description for event_d=%s, error %d",
                  event_id,GetLastError());
  }
/*
  结果:
  德国事件:50
              [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importa
   [ 0] 276010001      1        6           2           0          276      1
   [ 1] 276010002      1        6           2           0          276      1
   [ 2] 276010003      1        4           2           0          276      1
   [ 3] 276010004      1        4           2           0          276      1
   ....
   [47] 276500001      1        8           2           0          276      0
   [48] 276500002      1        8           2           0          276      0
   [49] 276500003      1        8           2           0          276      0
  接收event_id=276500003事件描述
  国家:德国 国家代码= 276)
  事件名称:Markit综合PMI
  事件代码:markit-composite-pmi
   事件重要性: CALENDAR_IMPORTANCE_MODERATE
   事件类型:CALENDAR_TYPE_INDICATOR
   事件版块:CALENDAR_SECTOR_BUSINESS
   事件频率:CALENDAR_FREQUENCY_MONTH
   事件发布模式: CALENDAR_TIMEMODE_DATETIME
   事件计量单位:CALENDAR_UNIT_NONE
  小数位数:1
   乘数值:CALENDAR_MULTIPLIER_NONE
   源URL:https://www.markiteconomics.com
```

© 2000-2025, MetaQuotes Ltd.

```
*/
```

另 见

CalendarEventByCountry, CalendarEventByCurrency, CalendarValueById

© 2000-2025, MetaQuotes Ltd.

# CalendarValueById

根据ID获得事件值描述。

```
bool  CalendarValueById(
   ulong                value_id,      // 事件值ID
   MqlCalendarValue&    value          // 用于接收事件值的变量
   );
```

参数

*value_id*

[in] 事件值ID。

*值*

[out] MqlCalendarValue类型变量，用于接收事件描述。

返回值

如果成功返回true，否则返回false。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 5402 -ERR_CALENDAR_NO_DATA (国家未找到)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。

注意

所有用于经济日历的函数都使用交易服务器时间(TimeTradeServer)。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

MqlCalendarValue结构提供了检查和设置actual_value、forecast_value、prev_value和revised_prev_value字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN (-9223372036854775808)。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 日本国家代码(ISO 3166-1 Alpha-2)
   string japan_code="JP";
//--- 设置事件发生间隔的边界
   datetime date_from=D'01.01.2018';  // 获取2018年的所有事件
   datetime date_to=0;                // 0 指所有已知事件，包括尚未发生的事件
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 获得日本事件值的数组
   MqlCalendarValue values[];
   int values_count=CalendarValueHistory(values,date_from,date_to,japan_code);
//--- 沿着检测到的事件值移动
   if(values_count>0)
     {
      PrintFormat("Number of values for Japan events: %d",values_count);
      //--- 删除所有 "空"值(actual_value==-9223372036854775808)
      for(int i=values_count-1;i>=0;i--)
        {
         if(values[i].actual_value==-9223372036854775808)
            ArrayRemove(values,i,1);
        }
      PrintFormat("Number of values after deleting empty ones: %d",ArraySize(values));
     }
   else
     {
      PrintFormat("Failed to receive events for the country code %s, error %d",
                  japan_code,GetLastError());
      //--- 脚本提前完成
      return;
     }
//--- 在values[]数组中保留不超过10个值
   if(ArraySize(values)>10)
     {
      PrintFormat("Reduce the list of values to 10 and display them");
      ArrayRemove(values,0,ArraySize(values)-10);
     }
   ArrayPrint(values);

//--- 现在，让我们展示如何基于已知的value_id获得事件值描述
   for(int i=0;i<ArraySize(values);i++)
     {
      MqlCalendarValue value;
      CalendarValueById(values[i].id,value);
      PrintFormat("%d: value_id=%d value=%d impact=%s",
                  i,values[i].id,value.actual_value,EnumToString(ENUM_CALENDAR_EVENT_
     }
//---
  }
/*
  结果：
  日本事件值的数量:1734
  删除空值后的值数量:1017
  将值列表减少到10并显示
        [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 56500  392030004 2019.03.28 23:30:00 2019.03.01 00:00:00          0          90
   [1] 56501  392030005 2019.03.28 23:30:00 2019.03.01 00:00:00          0          70
   [2] 56502  392030006 2019.03.28 23:30:00 2019.03.01 00:00:00          0        1100
```

© 2000-2025, MetaQuotes Ltd.

```
   [3] 56544  392030007 2019.03.28 23:30:00 2019.02.01 00:00:00          0         230
   [4] 56556  392050002 2019.03.28 23:30:00 2019.02.01 00:00:00          0         163
   [5] 55887  392020003 2019.03.28 23:50:00 2019.02.01 00:00:00          0         40
   [6] 55888  392020004 2019.03.28 23:50:00 2019.02.01 00:00:00          0        -180
   [7] 55889  392020002 2019.03.28 23:50:00 2019.02.01 00:00:00          0         20
   [8] 55948  392020006 2019.03.28 23:50:00 2019.02.01 00:00:00          1         140
   [9] 55949  392020007 2019.03.28 23:50:00 2019.02.01 00:00:00          1        -100
  显示基于value_id的关于事件值的简要数据
   0: value_id=56500 value=900000 impact=CALENDAR_IMPACT_POSITIVE
   1: value_id=56501 value=700000 impact=CALENDAR_IMPACT_NA
   2: value_id=56502 value=1100000 impact=CALENDAR_IMPACT_POSITIVE
   3: value_id=56544 value=2300000 impact=CALENDAR_IMPACT_NEGATIVE
   4: value_id=56556 value=1630000 impact=CALENDAR_IMPACT_POSITIVE
   5: value_id=55887 value=400000 impact=CALENDAR_IMPACT_NEGATIVE
   6: value_id=55888 value=-1800000 impact=CALENDAR_IMPACT_POSITIVE
   7: value_id=55889 value=200000 impact=CALENDAR_IMPACT_NEGATIVE
   8: value_id=55948 value=1400000 impact=CALENDAR_IMPACT_POSITIVE
   9: value_id=55949 value=-1000000 impact=CALENDAR_IMPACT_NEGATIVE
*/
```

另见

CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent, CalendarValueLast

© 2000-2025, MetaQuotes Ltd.

# CalendarCountries

获得“日历”中可用的国家名称数组。

```
int  CalendarCountries(
   MqlCalendarCountry&  countries[]          // 用于接收“日历”国家描述列表的数组
   );
```

参数

*countries[]*

[out] MqlCalendarCountry类型数组，用于接收所有“日历”国家的描述。

返回值

所接收的描述数量。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR（一般运行时错误）。
- 5401 -ERR_CALENDAR_TIMEOUT（超过请求时限）。
- 5400 -ERR_CALENDAR_MORE_DATA（数组大小不足以接收所有国家的描述，只可以接收那些适合的国家）。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 从经济日历中获得国家列表
   MqlCalendarCountry countries[];
   int count=CalendarCountries(countries);
//--- 在“日志”中显示数组
   if(count>0)
      ArrayPrint(countries);
   else
      PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
/*
  结果：
        [id]            [name] [code] [currency] [currency_symbol]       [url_name] [re
   [ 0]    0 "Worldwide"       "WW"   "ALL"      ""                 "worldwide"
   [ 1]  999 "European Union"  "EU"   "EUR"      "€"                "european-union"
   [ 2]  840 "United States"   "US"   "USD"      "$"                "united-states"
   [ 3]  124 "Canada"          "CA"   "CAD"      "$"                "canada"
   [ 4]   36 "Australia"       "AU"   "AUD"      "$"                "australia"
   [ 5]  554 "New Zealand"     "NZ"   "NZD"      "$"                "new-zealand"
   [ 6]  392 "Japan"           "JP"   "JPY"      "Ґ"                "japan"
   [ 7]  156 "China"           "CN"   "CNY"      "Ґ"                "china"
   [ 8]  826 "United Kingdom"  "GB"   "GBP"      "J"                "united-kingdom"
   [ 9]  756 "Switzerland"     "CH"   "CHF"      "₣"                "switzerland"
   [10]  276 "Germany"         "DE"   "EUR"      "€"                "germany"
```

© 2000-2025, MetaQuotes Ltd.

```
   [11]  250 "France"         "FR"   "EUR"      "€"              "france"
   [12]  380 "Italy"          "IT"   "EUR"      "€"              "italy"
   [13]  724 "Spain"          "ES"   "EUR"      "€"              "spain"
   [14]   76 "Brazil"         "BR"   "BRL"      "R$"             "brazil"
   [15]  410 "South Korea"    "KR"   "KRW"      "₩"              "south-korea"
*/
  }
```

另 见

CalendarEventByCountry, CalendarCountryById

© 2000-2025, MetaQuotes Ltd.

# CalendarEventByCountry

根据指定国家代码获得日历中可用的所有事件描述数组。

```
int  CalendarEventByCountry(
   string               country_code,     // 国家代码名称(ISO 3166-1 alpha-2)
   MqlCalendarEvent&    events[]          //用于接收描述数组的变量
   );
```

参数

*country_code*

[in] 国家代码名称(ISO 3166-1 alpha-2)

*events[]*

[out] MqlCalendarEvent类型数组，用于接收指定国家的所有事件描述。

返回值

所接收的描述数量。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 4004 -ERR_NOT_ENOUGH_MEMORY (内存不足，无法执行请求)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。
- 执行 ArrayResize()失败的错误

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 欧盟国家代码(ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- 获得欧盟事件
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(EU_code,events);
//--- 在“日志”中显示欧盟事件
   if(events_count>0)
     {
      PrintFormat("EU events: %d",events_count);
      ArrayPrint(events);
     }
//---
  }
/*
  结果:
  欧盟事件:56
             [id] [type]  [country_id] [unit] [importance] [multiplier] [digits] [ever
   [ 0] 999010001      0           999      0            2            0        0  "ECB
   [ 1] 999010002      0           999      0            2            0        0  "ECB
```

© 2000-2025, MetaQuotes Ltd.

```
   [ 2] 999010003      0          999      0          3          0          0  "ECB
   [ 3] 999010004      0          999      0          3          0          0  "ECB
   [ 4] 999010005      0          999      0          2          0          0  "ECB
   [ 5] 999010006      1          999      1          3          0          2  "ECB
   [ 6] 999010007      1          999      1          3          0          2  "ECB
   [ 7] 999010008      0          999      0          2          0          0  "ECB
   [ 8] 999010009      1          999      2          2          3          3  "ECB
   [ 9] 999010010      0          999      0          2          0          0  "ECB
   [10] 999010011      0          999      0          2          0          0  "ECB
   ...

*/
```

另见

CalendarCountries, CalendarCountryById

© 2000-2025, MetaQuotes Ltd.

# CalendarEventByCurrency

根据指定货币获得日历中可用的所有事件描述数组。

```
int  CalendarEventByCurrency(
   const string         currency,      // 国家货币代码名称
   MqlCalendarEvent&    events[]       // 用于接收描述数组的变量
   );
```

参数

*currency*

[in] 国家货币代码名称。

*events[]*

[out] MqlCalendarEvent类型数组，用于接收指定货币的所有事件描述。

返回值

所接收的描述数量。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 4004 -ERR_NOT_ENOUGH_MEMORY (内存不足，无法执行请求)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。
- 执行 ArrayResize()失败的错误

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明用于接收经济日历事件的数组
      MqlCalendarEvent events[];
//--- 获得欧盟货币事件
      int count = CalendarEventByCurrency("EUR",events);
      Print("count = ", count);
//--- 10个事件足以满足当前示例
      if(count>10)
         ArrayResize(events,10);
//--- 在“日志”中显示事件
      ArrayPrint(events);
  }
/*
  结果：
             [id] [type] [country_id] [unit] [importance]
   [0] 999010001      0          999      0            2 "https://www.ecb.europa.eu/ho
   [1] 999010002      0          999      0            2 "https://www.ecb.europa.eu/ho
   [2] 999010003      0          999      0            3 "https://www.ecb.europa.eu/ho
   [3] 999010004      0          999      0            3 "https://www.ecb.europa.eu/ho
```

© 2000-2025, MetaQuotes Ltd.

```
   [4] 999010005      0          999      0           2 "https://www.ecb.europa.eu/hc
   [5] 999010006      1          999      1           3 "https://www.ecb.europa.eu/hc
   [6] 999010007      1          999      1           3 "https://www.ecb.europa.eu/hc
   [7] 999010008      0          999      0           2 "https://www.ecb.europa.eu/hc
   [8] 999010009      1          999      2           2 "https://www.ecb.europa.eu/hc
   [9] 999010010      0          999      0           2 "https://www.ecb.europa.eu/hc
*/
```

另见

CalendarEventById, CalendarEventByCountry

© 2000-2025, MetaQuotes Ltd.

# CalendarValueHistoryByEvent

根据事件ID获得指定时期的所有事件值数组。

```
bool  CalendarValueHistoryByEvent(
   ulong              event_id,          // 事件ID
   MqlCalendarValue&  values[],          // 值描述数组
   datetime           datetime_from,     // 时间范围的左边框
   datetime           datetime_to=0      // 时间范围的右边框
   );
```

参数

*event_id*

[in] 事件ID。

*values[]*

[out] MqlCalendarValue类型数组，用于接收事件值。

*datetime_from*

[in] 时间范围事件的初始日期通过指定ID选择，而*datetime_from* < *datetime_to*。

*datetime_to=0*

[in] 时间范围事件的结束日期通过指定ID选择。如果*datetime_to*没有设置（或设置为0），那么返回在"日历"数据库中从指定*datetime_from*日期开始的所有事件值（包括未来事件的值）。

返回值

如果成功返回true，否则返回false。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 - ERR_INTERNAL_ERROR（一般运行时错误）。
- 4004 - ERR_NOT_ENOUGH_MEMORY（内存不足，无法执行请求）。
- 5401 - ERR_CALENDAR_TIMEOUT（超过请求时限）。
- 5400 - ERR_CALENDAR_MORE_DATA（数组大小不足以接收所有值的描述，只可以接收那些适合的值）。
- 执行 ArrayResize()失败的错误

注意

所有用于经济日历的函数都使用交易服务器时间（TimeTradeServer）。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

MqlCalendarValue结构提供了检查和设置actual_value、forecast_value、prev_value和revised_prev_value字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN（-9223372036854775808）。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

© 2000-2025, MetaQuotes Ltd.

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 欧盟国家代码(ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- 获得欧盟事件
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(EU_code,events);
//--- 在“日志”中显示欧盟事件
   if(events_count>0)
     {
      PrintFormat("EU events: %d",events_count);
      //--- 减少事件列表，10个事件足以分析
      ArrayResize(events,10);
      ArrayPrint(events);
     }
//--- 查阅“ECB利率决策”事件event_id=999010007
   ulong event_id=events[6].id;         // 事件ID可在“日历”中更改，因此一定进行验证
   string event_name=events[6].name;    // “日历”事件的名称
   PrintFormat("Get values for event_name=%s event_id=%d",event_name,event_id);
//--- 获得“ECB利率决策”事件的所有值
   MqlCalendarValue values[];
//--- 设置事件发生间隔的边界
   datetime date_from=0;            // 获取可用历史开始的所有事件
   datetime date_to=D'01.01.2016'; // 获取2016年以后的事件
   if(CalendarValueHistoryByEvent(event_id,values,date_from,date_to))
     {
      PrintFormat("Received values for %s: %d",
                  event_name,ArraySize(values));
      //--- 减少值列表，10个事件足以分析
      ArrayResize(values,10);
      ArrayPrint(values);
     }
   else
     {
      PrintFormat("Error! Failed to get values for event_id=%d",event_id);
      PrintFormat("Error code: %d",GetLastError());
     }
  }
//---
/*
  结果：
  欧盟事件:56
             [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importar
   [0] 999010001      0        5           0           0          999      0
```

© 2000-2025, MetaQuotes Ltd.

```
   [1] 999010002      0        5           0           0          999      0
   [2] 999010003      0        5           0           0          999      0
   [3] 999010004      0        5           0           0          999      0
   [4] 999010005      0        5           0           0          999      0
   [5] 999010006      1        5           0           0          999      1
   [6] 999010007      1        5           0           0          999      1
   [7] 999010008      0        5           0           0          999      0
   [8] 999010009      1        5           0           0          999      2
   [9] 999010010      0        5           0           0          999      0
  获得event_name=ECB Interest Rate Decision值 event_id=999010007
  已接收的 "ECB利率决策"事件值:102
       [id] [event_id]              [time]             [period] [revision] [actual_valu
   [0] 2776  999010007 2007.03.08 11:45:00 1970.01.01 00:00:00          0         3750(
   [1] 2777  999010007 2007.05.10 11:45:00 1970.01.01 00:00:00          0         3750(
   [2] 2778  999010007 2007.06.06 11:45:00 1970.01.01 00:00:00          0         4000(
   [3] 2779  999010007 2007.07.05 11:45:00 1970.01.01 00:00:00          0         4000(
   [4] 2780  999010007 2007.08.02 11:45:00 1970.01.01 00:00:00          0         4000(
   [5] 2781  999010007 2007.09.06 11:45:00 1970.01.01 00:00:00          0         4000(
   [6] 2782  999010007 2007.10.04 11:45:00 1970.01.01 00:00:00          0         4000(
   [7] 2783  999010007 2007.11.08 12:45:00 1970.01.01 00:00:00          0         4000(
   [8] 2784  999010007 2007.12.06 12:45:00 1970.01.01 00:00:00          0         4000(
   [9] 2785  999010007 2008.01.10 12:45:00 1970.01.01 00:00:00          0         4000(
*/
```

另见

CalendarCountries, CalendarEventByCountry, CalendarValueHistory, CalendarEventById, CalendarValueById

© 2000-2025, MetaQuotes Ltd.

# CalendarValueHistory

通过根据国家和/或货币进行排序的能力，获得指定时期的所有事件值数组。

```
bool  CalendarValueHistory(
   MqlCalendarValue&  values[],              // 值描述数组
   datetime           datetime_from,         // 时间范围的左边框
   datetime           datetime_to=0          //时间范围的右边框
   const string       country_code=NULL,     // 国家代码名称(ISO 3166-1 alpha-2)
   const string       currency=NULL          // 国家货币代码名称
   );
```

参数

*values[]*

[out] MqlCalendarValue类型数组，用于接收事件值。

*datetime_from*

[in] 时间范围事件的初始日期通过指定ID选择，而*datetime_from* < *datetime_to*。

*datetime_to=0*

[in] 时间范围事件的结束日期通过指定ID选择。如果*datetime_to* 没有设置（或设置为0），那么返回在“日历”数据库中从指定*datetime_from*日期开始的所有事件值（包括未来事件的值）。

*country_code=NULL*

[in] 国家代码名称(ISO 3166-1 alpha-2)

*currency=NULL*

[in] 国家货币代码名称。

返回值

如果成功返回true，否则返回false。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR（一般运行时错误）。
- 4004 -ERR_NOT_ENOUGH_MEMORY（内存不足，无法执行请求）。
- 5401 -ERR_CALENDAR_TIMEOUT（超过请求时限）。
- 5400 -ERR_CALENDAR_MORE_DATA（数组大小不足以接收所有值的描述，只可以接收那些适合的值）。
- 执行ArrayResize()失败的错误

注意

所有用于经济日历的函数都使用交易服务器时间(TimeTradeServer)。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

如果固定长度的*events[]*数组被传递到函数且没有足够空间保存整个结果，则会激活ERR_CALENDAR_MORE_DATA (5400)错误。

© 2000-2025, MetaQuotes Ltd.

如果*datetime_to*没有设置（或设置为0），那么返回在“日历”数据库中从指定*datetime_from*日期开始的所有事件值（包括未来事件的值）。

对于*country_code*和*currency*过滤器，NULL和""值是相同的，表示没有过滤器。

对于*country_code*，MqlCalendarCountry结构的*code*字段，例如应使用"US"、"RU"或"EU"。

对于*currency*，MqlCalendarCountry结构的*currency*字段，例如应使用"USD"、"RUB"或"EUR"。

过滤器是通过连接使用的，即合理'AND'仅用于选择同时满足两个条件（国家和货币）的事件值。

MqlCalendarValue结构提供了检查和设置actual_value、forecast_value、prev_value和revised_prev_value字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN (-9223372036854775808)。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 欧盟国家代码(ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- 获得所有欧盟事件值
   MqlCalendarValue values[];
//--- 设置事件发生间隔的边界
   datetime date_from=D'01.01.2018';  // 获取2018年的所有事件
   datetime date_to=0;                // 0 指所有已知事件，包括尚未发生的事件
//--- 请求2018年以来的欧盟事件历史
   if(CalendarValueHistory(values,date_from,date_to,EU_code))
     {
      PrintFormat("Received event values for country_code=%s: %d",
                  EU_code,ArraySize(values));
      //--- 减少输出到“日志”的数组大小
      ArrayResize(values,10);
//--- 在“日志”中显示事件值
      ArrayPrint(values);
     }
   else
     {
      PrintFormat("Error! Failed to receive events for country_code=%s",EU_code);
      PrintFormat("Error code: %d",GetLastError());
     }
//---
```

© 2000-2025, MetaQuotes Ltd.

```
  }
/*
  结果:
  已接收的country_code=EU事件值:1384
        [id] [event_id]           [time]               [period] [revision]    [actual_v
  [0] 54215  999500001 2018.01.02 09:00:00 2017.12.01 00:00:00           3        60600
  [1] 54221  999500002 2018.01.04 09:00:00 2017.12.01 00:00:00           3        56600
  [2] 54222  999500003 2018.01.04 09:00:00 2017.12.01 00:00:00           3        58100
  [3] 45123  999030005 2018.01.05 10:00:00 2017.11.01 00:00:00           0          600
  [4] 45124  999030006 2018.01.05 10:00:00 2017.11.01 00:00:00           0         2800
  [5] 45125  999030012 2018.01.05 10:00:00 2017.12.01 00:00:00           1          900
  [6] 45126  999030013 2018.01.05 10:00:00 2017.12.01 00:00:00           1         1400
  [7] 54953  999520001 2018.01.05 20:30:00 2018.01.02 00:00:00           0       127900
  [8] 22230  999040003 2018.01.08 10:00:00 2017.12.01 00:00:00           0         9100
  [9] 22231  999040004 2018.01.08 10:00:00 2017.12.01 00:00:00           0        18400
*/
```

另 见

CalendarCountries, CalendarEventByCountry, CalendarValueHistoryByEvent, CalendarEventById, CalendarValueById

© 2000-2025, MetaQuotes Ltd.

# CalendarValueLastByEvent

通过指定change_id，从日历数据库状态根据ID获得事件值数组。

```
int  CalendarValueLastByEvent(
   ulong                event_id,      // 事件ID
   ulong&               change_id,     // 事件值ID
   MqlCalendarValue&    values[]       // 值描述数组
   );
```

参数

*event_id*

[in] 事件ID。

*change_id*

[in][out] 更改ID。

*values[]*

[out] MqlCalendarValue类型数组，用于接收事件值。

返回值

接收事件值的数量。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 4004 -ERR_NOT_ENOUGH_MEMORY (内存不足，无法执行请求)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。
- 5400 -ERR_CALENDAR_MORE_DATA (数组大小不足以接收所有值的描述，只可以接收那些适合的值)。
- 执行 ArrayResize()失败的错误

注意

所有用于经济日历的函数都使用交易服务器时间(TimeTradeServer)。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

如果固定长度的*events[]*数组被传递到函数且没有足够空间保存整个结果，则会激活ERR_CALENDAR_MORE_DATA (5400)错误。

如果*change_id* = 0被传递到该函数，那么该函数会始终返回零值，但当前日历数据库返回到*change_id*。

该函数返回指定新闻的数组和可用于函数后续调用来接收新闻新值的新*change_id*。因此，可以通过最后已知的*change_id*调用这个函数来更新指定新闻的值。

MqlCalendarValue结构提供了检查和设置actual_value、forecast_value、prev_value和revised_prev_value字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN (-9223372036854775808)。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定

© 2000-2025, MetaQuotes Ltd.

一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

EA听取非农就业人口报告发布的样本：

```
#property description "Example of using the CalendarValueLastByEvent function"
#property description " for tracking the release of the Nonfarm Payrolls report."
#property description "To achieve this, get the current change ID"
#property description " of the Calendar database. Then, use this ID to receive"
#property description " only new events via the timer survey"
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建计时器
   EventSetTimer(60);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 销毁计时器
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer函数                                                        |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- 日历数据库更改ID
   static ulong calendar_change_id=0;
//--- 第一次启动属性
   static bool first=true;
//--- 事件ID
   static ulong event_id=0;
//--- 事件名称
   static string event_name=NULL;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 事件值数组
   MqlCalendarValue values[];
//--- 执行初始化 - 获得当前calendar_change_id
   if(first)
     {
      MqlCalendarEvent events[];
      //--- 美国国家代码(ISO 3166-1 Alpha-2)
      string USA_code="US";
      //--- 获得美国事件
      int events_count=CalendarEventByCountry(USA_code,events);
      //--- 'events'数组中必要事件的位置
      int event_pos=-1;
      //--- 在"日志"中显示美国事件
      if(events_count>0)
        {
         PrintFormat("%s: USA events: %d",__FUNCTION__,events_count);
         for(int i=0;i<events_count;i++)
           {
            string event_name_low=events[i].name;
            //--- 将事件名称更改为小写
            if(!StringToLower(event_name_low))
              {
               PrintFormat("StringToLower() returned %d error",GetLastError());
               //--- 提前退出函数
               return;
              }
            //--- 寻找"非农就业人口"事件
            if(StringFind(event_name_low,"nonfarm payrolls")!=-1)
              {
               //--- 事件已找到,请记住其ID
               event_id=events[i].id;
               //--- 书写"非农就业人口"事件名称
               event_name=events[i].name;
               //--- 记住'events[]'数组中事件的位置
               event_pos=i;
               //--- 请记住,该"日历"中包含多个名称中有"非农就业人口"的事件
               PrintFormat("Event \"Nonfarm Payrolls\" found: event_id=%d  event_name=
               //--- 通过注释'break'操作符来查看所有事件,以便更好地理解这个示例
               break;
              }
           }
         //--- 通过删除"非农就业人口"之后的事件来减少列表
         ArrayRemove(events,event_pos+1);
         //--- 将9个事件放在"非农就业人口"之前,以便分析更方便
         ArrayRemove(events,0,event_pos-9);
         ArrayPrint(events);
        }
      else
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         PrintFormat("%s: CalendarEventByCountry(%s) returned 0 events, error code=%d"
                     USA_code,__FUNCTION__,GetLastError());
         //--- 操作完成失败，请在下一次调用计时器时重试
         return;
        }

      //--- 获得指定事件的 "日历"数据库更改ID
      if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
        {
         //--- 第一次启动时不可执行这个代码块，但可以添加它
         PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                     __FUNCTION__,calendar_change_id);
         //--- 设置该标识并在计时器的下一个事件之前退出
         first=false;
         return;
        }
      else
        {
         //--- 无法接收数据 对于第一次启动，这是正常的)，请检查是否有错误
         int error_code=GetLastError();
         if(error_code==0)
           {
            PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                        __FUNCTION__,calendar_change_id);
            //--- 设置该标识并在计时器的下一个事件之前退出
            first=false;
            //--- 现在我们有calendar_change_id值
            return;
           }
         else
           {
            //--- 并且这确实是一个错误
            PrintFormat("%s: Failed to get values for event_id=%d",__FUNCTION__,event_
            PrintFormat("Error code: %d",error_code);
            //--- 操作完成失败，请在下一次调用计时器时重试
            return;
           }
        }
     }

//--- 我们有 "日历"更改ID(change_id)的最后知道的值
   ulong old_change_id=calendar_change_id;
//--- 请检查新的 "非农就业人口"事件值
   if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
     {
      PrintFormat("%s: Received new events for \"%s\": %d",
                  __FUNCTION__,event_name,ArraySize(values));
      //--- 在 "日志"中显示'value'数组的数据
      ArrayPrint(values);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 在“日志”中显示之前和新的“日历ID”的值
      PrintFormat("%s: Previous change_id=%d, new change_id=%d",
                  __FUNCTION__,old_change_id,calendar_change_id);
/*
      编写您将在这里处理“非农就业人口”数据发布的代码
      */
     }
//---
  }
/*
  结果：
   OnTimer: USA events: 202
  Event "Nonfarm Payrolls" found: event_id=840030016  event_name=Nonfarm Payrolls
            [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importar
   [0] 840030007      1        4           2           0           840      1
   [1] 840030008      1        4           2           0           840      1
   [2] 840030009      1        4           2           0           840      0
   [3] 840030010      1        4           2           0           840      0
   [4] 840030011      1        4           2           0           840      1
   [5] 840030012      1        4           2           0           840      1
   [6] 840030013      1        4           2           0           840      1
   [7] 840030014      1        4           2           0           840      1
   [8] 840030015      1        3           2           0           840      1
   [9] 840030016      1        3           2           0           840      4
   OnTimer: Received the Calendar database current ID: change_id=33986560

*/
```

另见

CalendarValueLast, CalendarValueHistory, CalendarValueHistoryByEvent, CalendarValueById

© 2000-2025, MetaQuotes Ltd.

# CalendarValueLast

通过指定change_id和根据国家和/或货币进行排序的能力，从日历数据库状态获得所有事件值数组。

```
int  CalendarValueLast(
   ulong&               change_id,              // 更改ID
   MqlCalendarValue&    values[],               // 值描述数组
   const string         country_code=NULL,      // 国家代码名称(ISO 3166-1 alpha-2)
   const string         currency=NULL           // 国家货币代码名称
   );
```

参数

*change_id*

[in][out] 更改ID。

*values[]*

[out] MqlCalendarValue类型数组，用于接收事件值。

*country_code=NULL*

[in] 国家代码名称(ISO 3166-1 alpha-2)

*currency=NULL*

[in] 国家货币代码名称。

返回值

接收事件值的数量。若要获得有关错误的信息，请调用GetLastError()函数。可能错误：

- 4001 -ERR_INTERNAL_ERROR (一般运行时错误)。
- 4004 -ERR_NOT_ENOUGH_MEMORY (内存不足，无法执行请求)。
- 5401 -ERR_CALENDAR_TIMEOUT (超过请求时限)。
- 5400 -ERR_CALENDAR_MORE_DATA (数组大小不足以接收所有值的描述，只可以接收那些适合的值)。
- 执行ArrayResize()失败的错误

注意

所有用于经济日历的函数都使用交易服务器时间(TimeTradeServer)。这意味着MqlCalendarValue结构中的时间和CalendarValueHistoryByEvent/CalendarValueHistory函数中输入的时间都在交易服务器时区中设置，而不是在用户本地时间设置。

如果固定长度的*events[]*数组被传递到函数且没有足够空间保存整个结果，则会激活ERR_CALENDAR_MORE_DATA (5400)错误。

如果*change_id* = 0被传递到该函数，那么您将获得这个参数日历数据库的当前*change_id*；该函数返回0

对于*country_code*和*currency*过滤器，NULL和""值是相同的，表示没有过滤器。

对于*country_code*，MqlCalendarCountry结构的*code*字段，例如应使用"US"、"RU"或"EU"。

对于*currency*，MqlCalendarCountry结构的*currency*字段，例如应使用"USD"、"RUB"或"EUR"。

© 2000-2025, MetaQuotes Ltd.

过滤器是通过连接使用的，即合理'AND'仅用于选择同时满足两个条件（国家和货币）的事件值

该函数返回指定新闻的数组和可用于函数后续调用来接收新闻新值的新*change_id*。因此，可以通过最后已知的*change_id*调用这个函数来更新指定新闻的值。

MqlCalendarValue结构提供了检查和设置actual_value、forecast_value、prev_value和revised_prev_value字段值的方法。如果没有指定任何值，则该字段存储LONG_MIN（-9223372036854775808）。

请注意，存储在这些字段中的值要乘以100万。这表示当您使用函数CalendarValueById、CalendarValueHistoryByEvent、CalendarValueHistory、CalendarValueLastByEvent、CalendarValueLast接收MqlCalendarValue中的值时，您应该检查该字段值是否等于LONG_MIN；如果在字段中指定一个值，那么您应该将该值除以1,000,000以获得该值。另一种获取该值的方法是使用MqlCalendarValue结构的函数检查和获取值。

监听经济日历事件的示例EA：

```
#property description "Example of using the CalendarValueLast function"
#property description " to develop the economic calendar events listener."
#property description "To achieve this, get the current change ID"
#property description " of the Calendar database. Then, use this ID to receive"
#property description " only new events via the timer survey"
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建计时器
   EventSetTimer(60);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 销毁计时器
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer函数                                                        |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- 日历数据库更改ID
   static ulong calendar_change_id=0;
//--- 第一次启动属性
   static bool first=true;
//--- 事件值数组
   MqlCalendarValue values[];
//--- 执行初始化 - 获得当前calendar_change_id
   if(first)
     {
      //--- get the Calendar database change ID
      if(CalendarValueLast(calendar_change_id,values)>0)
        {
         //--- 第一次启动时不可执行这个代码块，但可以添加它
         PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                     __FUNCTION__,calendar_change_id);
         //--- 设置该标识并在计时器的下一个事件之前退出
         first=false;
         return;
        }
      else
        {
         //--- 无法接收数据 对于第一次启动，这是正常的)，请检查是否有错误
         int error_code=GetLastError();
         if(error_code==0)
           {
            PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                        __FUNCTION__,calendar_change_id);
            //--- 设置该标识并在计时器的下一个事件之前退出
            first=false;
            //--- 现在我们有calendar_change_id值
            return;
           }
         else
           {
            //--- 并且这确实是一个错误
            PrintFormat("%s: Failed to get events in CalendarValueLast. Error code: %d",
                        __FUNCTION__,error_code);
            //--- 操作完成失败，请在下一次调用计时器时重新初始化
            return;
           }
        }
     }

//--- 我们有 "日历"更改ID(change_id)的最后知道的值
   ulong old_change_id=calendar_change_id;
//--- 检查是否有新的 "日历"事件
```

© 2000-2025, MetaQuotes Ltd.

```
   if(CalendarValueLast(calendar_change_id,values)>0)
     {
      PrintFormat("%s: Received new Calendar events: %d",
                  __FUNCTION__,ArraySize(values));
      //--- 在“日志”中显示'value'数组的数据
      ArrayPrint(values);
      //--- 在“日志”中显示之前和新的“日历ID”的值
      PrintFormat("%s: Previous change_id=%d, new change_id=%d",
                  __FUNCTION__,old_change_id,calendar_change_id);
      //--- 在“日志”中显示新事件
      ArrayPrint(values);
      /*
     编写您将在这里处理所发生事件的代码
      */
     }
//---
  }
/*
  监听操作示例:
  OnTimer:已接收的“日历数据库”,当前ID:change_id=33281792
  OnTimer:已接收的“日历”新事件:1
        [id] [event_id]              [time]            [period] [revision] [actual_val
   [0] 91040   76020013 2019.03.20 15:30:00 1970.01.01 00:00:00          0       -507
  OnTimer: Previous change_id=33281792, new change_id=33282048
        [id] [event_id]              [time]            [period] [revision] [actual_val
   [0] 91040   76020013 2019.03.20 15:30:00 1970.01.01 00:00:00          0       -507
  OnTimer:已接收的“日历”新事件:1
        [id] [event_id]              [time]            [period] [revision]       [actu
   [0] 91041   76020013 2019.03.27 15:30:00 1970.01.01 00:00:00          0 -922337203
  OnTimer: Previous change_id=33282048, new change_id=33282560
        [id] [event_id]              [time]            [period] [revision]       [actu
   [0] 91041   76020013 2019.03.27 15:30:00 1970.01.01 00:00:00          0 -922337203

*/
```

另见

CalendarValueLast, CalendarValueHistory, CalendarValueHistoryByEvent, CalendarValueById

© 2000-2025, MetaQuotes Ltd.

# 访问时间序列和指标数据

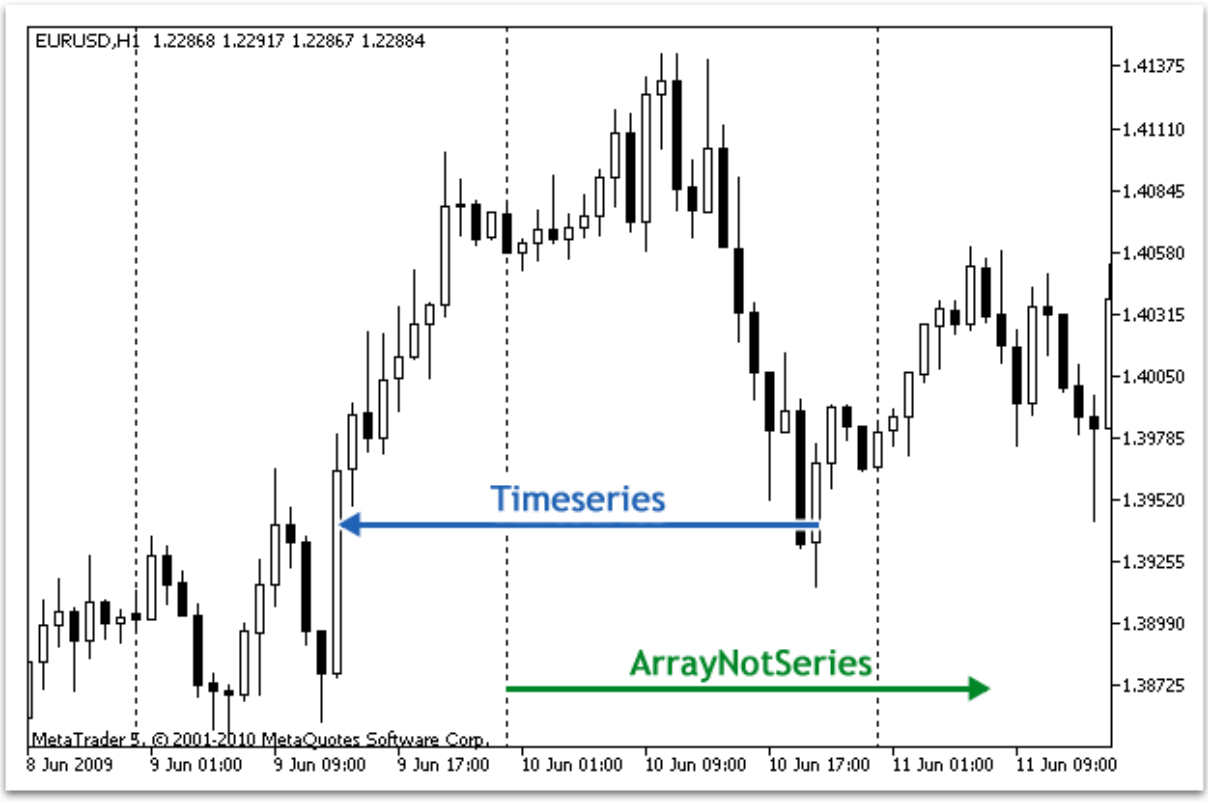
该组函数使用时序列和指标，时序列与通常的倒序的数据数组不同——时序列的元素从数组末尾到开头进行索引（大多从当前数据到最早数据），为了复制时序列值和指标数据，仅使用 动态数组 即可完成，因为复制函数为收到的值分配数组的必要大小。

这项规则有一个**重要的例外情况**：如果时序列和指标的值需要经常复制吗，例如，每次在EA交易中调用 OnTick() 或者在指标中调用 OnCalculate()，最好使用静态分配的数组，因为动态数组的内存分配操作需要额外时间，在测试和最佳化中会起到作用。

当使用函数访问时序列和指标的值时，要考虑索引方向，在 数组和时间序列中索引方向章节中有所描述。

指标和时序列数据的访问应用不考虑要求数据是否准备完好（也就是所谓的 异步访问），这对于自定义指标的计算来说非常重要，因此，如果没有数据，函数 *Copy...()* 类型会立即返回错误值。然而，当形成EA交易和脚本的访问时，接收数据的尝试会短暂停歇，只是为时序列要求的指标计算值的下载提供必要的时间。

组织访问数据 章节在MT5客户端中描述接收、存储和需求价格数据的细节。




历史上在数组中接收价格数据从末尾开始，物理上，新的数据通常从数组末尾开始编写，但数组索引通常是0,时序列数组中的0索引表示当前字节中的数据，也就是，在时间表中未完成的时间间隔。

时间表中单一价格字节可以形成，在 标准时间表 中预先定义21个。

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| SeriesInfoInteger | 返回历史资料信息 |
| Bars | 返回规定交易品种和周期的历史字节数量 |
| BarsCalculated | 错误情况下（还没有计算数据）返回指标缓冲区计算的数据量或者-1 |
| IndicatorCreate | 返回通过MqlParam参量类型数据创建的指定技术指标处理器 |
| IndicatorParameters | 基于指定的句柄，返回指标输入参数的数量，以及参数的值和类型 |
| IndicatorRelease | 如果没有其他，移除指标处理器并释放指标计算模块 |
| CopyBuffer | 从指定指标获得特定缓冲区数据到数组中。 |
| CopyRates | 将指定交易品种 比率结构的历史数据和周期放在数组中 |
| CopySeries | 从MqlRates结构获取指定交易品种周期和指定数量的同步时间序列。数据被收入到指定的数组集中。元素从现在到过去倒数，这意味着起始位置等于0表示当前柱形图。 |
| CopyTime | 将指定交易品种 开盘时间柱的历史数据和周期放在数组中 |
| CopyOpen | 将指定交易品种 开盘价格柱的历史数据和周期放在数组中 |
| CopyHigh | 将指定交易品种最大价格柱的历史数据和周期放在数组中 |
| CopyLow | 将指定交易品种最小价格柱的历史数据和周期放在数组中 |
| CopyClose | 将指定交易品种收盘价格柱的历史数据和周期放在数组中 |
| CopyTickVolume | 指定交易品种订单交易量历史数据和周期放在数组中 |
| CopyRealVolume | 指定交易品种交易量历史数据和周期放在数组中 |
| CopySpread | 指定交易品种点差历史数据和周期放在数组中 |
| CopyTicks | 获得程序端为当前工作期积累的报价并写入数组 |
| CopyTicksRange | 该函数可以在指定日期范围内以MqlTick格式接收报价到ticks_array |
| iBars | 返回可用于历史记录的相应交易品种和周期的柱形图的数量 |
| iBarShift | 根据时间搜索柱形图。这个函数返回与指定时间对应的柱形图指数 |
| iClose | 返回对应的图表上柱形图的收盘价（通过'shift'参数表示） |
| iHigh | 返回对应的图表上柱形图的最高价（通过'shift'参数表示） |
| iHighest | 返回在对应图表上找到的最高值的指数（相对于当前柱形图的转移） |
| iLow | 返回对应的图表上柱形图的最低价（通过'shift'参数表示） |
| iLowest | 返回在对应图表上找到的最小值的指数（相对于当前柱形图的转移） |
| iOpen | 返回对应的图表上柱形图的开盘价（通过'shift'参数表示） |
| iTime | 返回对应的图表上柱形图的开盘时间（通过'shift'参数表示） |
| iTickVolume | 返回对应的图表上柱形图的报价量（通过'shift'参数表示） |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| iRealVolume | 返回对应的图表上柱形图的真实交易量（通过'shift'参数表示） |
| iVolume | 返回对应的图表上柱形图的报价量（通过'shift'参数表示） |
| iSpread | 返回对应的图表上柱形图的点差值（通过'shift'参数表示） |

尽管通过 ArraySetAsSeries() 函数可能像在时序列一样建立 数组 接入元素，数组元素存储的可能是同一个——只有索引方向不同，为证明这个事实，提供如下示例：

```
   datetime TimeAsSeries[];
//--- 如时间序列访问一样设置数组访问
   ArraySetAsSeries(TimeAsSeries,true);
   ResetLastError();
   int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
   if(copied<=0)
     {
      Print("The copy operation of the open time values for last 10 bars has failed");
      return;
     }
   Print("TimeCurrent =",TimeCurrent());
   Print("ArraySize(Time) =",ArraySize(TimeAsSeries));
   int size=ArraySize(TimeAsSeries);
   for(int i=0;i<size;i++)
     {
      Print("TimeAsSeries["+i+"] =",TimeAsSeries[i]);
     }

   datetime ArrayNotSeries[];
   ArraySetAsSeries(ArrayNotSeries,false);
   ResetLastError();
   copied=CopyTime(NULL,0,0,10,ArrayNotSeries);
   if(copied<=0)
     {
      Print("The copy operation of the open time values for last 10 bars has failed");
      return;
     }
   size=ArraySize(ArrayNotSeries);
   for(int i=size-1;i>=0;i--)
     {
      Print("ArrayNotSeries["+i+"] =",ArrayNotSeries[i]);
     }
```

结果我们可以如下输出：

```
TimeCurrent = 2009.06.11 14:16:23
ArraySize(Time) = 10
TimeAsSeries[0] = 2009.06.11 14:00:00
TimeAsSeries[1] = 2009.06.11 13:00:00
TimeAsSeries[2] = 2009.06.11 12:00:00
```

© 2000-2025, MetaQuotes Ltd.

```
TimeAsSeries[3] = 2009.06.11 11:00:00
TimeAsSeries[4] = 2009.06.11 10:00:00
TimeAsSeries[5] = 2009.06.11 09:00:00
TimeAsSeries[6] = 2009.06.11 08:00:00
TimeAsSeries[7] = 2009.06.11 07:00:00
TimeAsSeries[8] = 2009.06.11 06:00:00
TimeAsSeries[9] = 2009.06.11 05:00:00

ArrayNotSeries[9] = 2009.06.11 14:00:00
ArrayNotSeries[8] = 2009.06.11 13:00:00
ArrayNotSeries[7] = 2009.06.11 12:00:00
ArrayNotSeries[6] = 2009.06.11 11:00:00
ArrayNotSeries[5] = 2009.06.11 10:00:00
ArrayNotSeries[4] = 2009.06.11 09:00:00
ArrayNotSeries[3] = 2009.06.11 08:00:00
ArrayNotSeries[2] = 2009.06.11 07:00:00
ArrayNotSeries[1] = 2009.06.11 06:00:00
ArrayNotSeries[0] = 2009.06.11 05:00:00
```

从结果中可以看到，TimeAsSeries数组增加的索引，索引减少的时间值，也就是说，从当前移到过去，对于普通数组ArrayNotSeries来说，结果是不同的——一旦索引增加，从过去移动到当前。

另见

数组动态，，ArrayGetAsSeries，ArraySetAsSeries，数据系列

© 2000-2025, MetaQuotes Ltd.

# 数组，缓冲器和时序列中索引方向

所有数组和指标缓冲区的默认索引是从左至右的，第一元素的索引等于0，因此，数组或指标缓冲区的第一元素默认从最左边的0开始，而最后一个元素在最右边。

指标缓冲区是双精度类型 动态数组 ，由客户端进行管理，因此相当于指标的字节数量。一个普通的双精度类型动态数组可以使用 SetIndexBuffer() 函数分派到指标缓冲区。指标缓冲区不要求使用 ArrayResize() 函数来建立大小-它会被终端的执行系统来执行。

时序列是颠倒搜索的数组，例如，时序列的第一个元素在最右边，最后一个元素在最左边。时序列是用来存储历史价格数据并包括时间信息，我们可以看见时序列最右边是最新数据，而最左边是最旧的数据。

因此时序列以0索引，包括最近引用的交易品种的信息。如果时序列包括每日时间列表的数据，当前尚未完成的日期会显示在0位置，而1索引位置会显示昨天的数据。

## 改变索引方向

函数 ArraySetAsSeries() 允许改变方位动态数组元素的方法，数据的自动命令在电脑内存中并未改变。该函数简单改变了安置数组元素的方法，因此当使用ArrayCopy()，函数复制一组数组到另一组中时，接收数组的命令会依据源数组的索引方向执行。

索引方向不能为分散的静态数组改变，即使数组以参量传递函数，想要改变内置的索引方向也是不可以的。

对于指标缓冲区来说，像普通数组一样，索引方向向后设置（像时序列），例如，关于指标缓冲区的零位置代表类似指标缓冲区的的最后值，也相当于指标最新字节的值。因此，指标的实际字节位置并未改变。

## 指标接收价格数据

每个自定义指标 都必要地建立 OnCalculate() 函数，价格数据需要传递指标缓冲区的计算值，搜索方向可以在传递数组中使用 ArrayGetAsSeries() 函数找到。

传递到函数的数组反应价格数据，例如，当检测数组时，这些数组都有时序列标志和函数 ArrayIsSeries()返回真值，然后，在许多情况下，索引方向应该只能通过 ArrayGetAsSeries()函数检测。

为了不依赖默认值，ArraySetAsSeries() 应该为数组无条件调用，并建立必要方向。

## 接收价格数据和指标值

在EA交易，指标和脚本中的所有默认检索方向都是从左至右。如果必要的话，在MQL5程序中，可以要求时序列价值是任意交易品种和时序列，指标的计算值是任意交易品种和时序列。

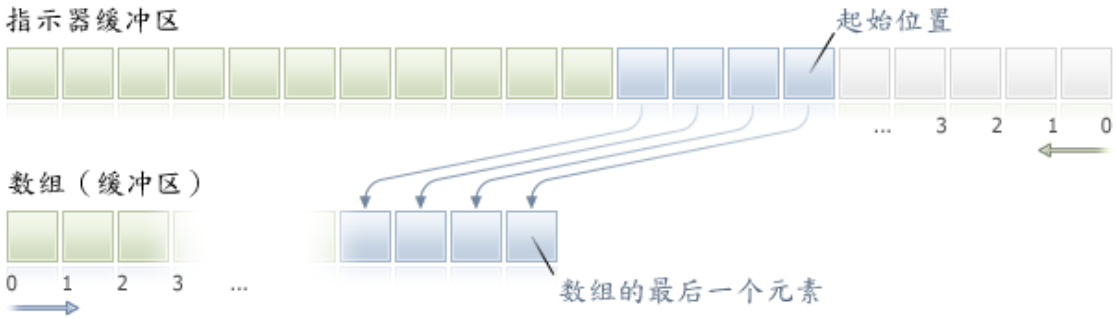
使用Copy...() 函数完成这些操作：

- 复制缓冲器 -复制指标缓冲区到双精度类型数组；
- 复制价格 -复制价格记录到结构 MqlRates类型数组；
- 复制时间 -复制时间值到日期时间类型数组；
- 复制开盘价 -复制开仓价格到双精度类型数组；
- 复制最高价 -复制最高价值到双精度类型数组；

© 2000-2025, MetaQuotes Ltd.

- 复制最低价 -复制最低价值到双精度类型数组；
- 复制收盘价 -复制收盘价格到双精度类型数组；
- 复制点成交量 -复制点成交量到长类型数组；
- 复制真实成交量 -复制空成交量到长类型数组；
- 复制点差 -复制点差到整型数组；

所有这些函数都以相同方式工作，考虑到CopyBuffer()在示例中包含的原理，暗示在需求数据的索引方向是时序列，0索引位置存储当前未完成字节的数据。为了房屋这些数据，需要复制必要的数据成交量容器数组，例如，数组 *buffer*。




当复制时，我们需要在源数组中指定起始位置，从数据复制的容器数组开始。在成功的条件下，指定元素数量从源数组中复制容器数组（假设从指标缓冲区中）。不考虑设置在容器数组中的索引值，复制执行总是显示在上述计算中。

如果预计价格数据将被大量迭代循环处理，那么您使用IsStopped()函数检查迫使程序终止这个事实更明智：

```
int copied=CopyBuffer(ma_handle,// 指标处理
                      0,        // 指标缓冲区标引
                      0,        // 复制的开始位置
                      number,   // 复制值的数量
                      Buffer    // 接收值的数组
                      );
if(copied<0) return;
int k=0;
while(k<copied && !IsStopped())
  {
   //--- 获得k标引的值
   double value=Buffer[k];
   // ...
   // 使用值
   k++;
  }
```

示例：

```
input int per=10; // 指数周期
int ma_handle;    // 指标处理
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   ma_handle=iMA(_Symbol,0,per,0,MODE_EMA,PRICE_CLOSE);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   double ema[10];
   int copied=CopyBuffer(ma_handle,// 指标处理
                         0,        // 指标缓冲区指数
                         0,        // 复制的初始位置
                         10,       // 用于复制的值数
                         ema       // 接收数组的值
                         );
   if(copied<0) return;
// .... 深层代码
  }
```

另见

组织数据存储

© 2000-2025, MetaQuotes Ltd.

# 组织数据存储

这一章节是讲述获得、存储和索取价格数据的（时间序列）。

## 从交易服务器中获得数据

在MT5客户端的价格数据执行之前，必须先收到然后经过处理，为了接收数据，要确定MT5交易服务器已经确定连接，数据根据终端要求以微小字节形式接收。

服务器的机械装置是参考数据的，并不依靠最初的要求-使用者通过操作图表或者MQL5语言程序方式。

## 储存媒介数据

从服务器重接收的数据自动打开或存储在HCC媒介信息中，每个象征数据都以分类文件夹 *terminal_directory*\bases*server_name*\history*symbol_name*的形式记录。例如，关于EURUSD的数据从MetaQuotes-Demo服务器中接收，但会存储在 *terminal_directory*\bases\MetaQuotes-Demo\history\EURUSD\中。

数据以.hcc的扩展名方式书写到文件中，每个文件存储分钟字节型数据一年，例如，2009年的数据文件包含2009年所有的分钟字节型数据，这些文件为所有时间架框提供价格数据的，他们并不能直接接入。

## 在媒介数据外获得必要的时间架构

媒介HCC文件以源数据类型使用，它在HC板式中以建立价格数据的要求来使用的，HC板式的数据是事件序列为快速接入的最大化准备，他们根据图表或MQL5程序的要求来构建，数据的成交量不能超过图表字节的最大值，数据长期存储使用hc扩展文件。

为了节约资源，在时间架构里的数据如果需要的话以ram格式存储，如果不能长时间调用，会从RAM中释放并保存到文件夹中，对于每个时间序列来说，数据的存贮无论是否已经有为另一时间架构准备的备用数据，形式规则和接入数据对于所有时间架构来说都是一样的。那就意味着，尽管在HCC中存储了一分钟数据，HCC数据的可行性并不意味着在同成交量中M1时间架构的可行性。

从服务器中接收新数据会调用所有时间架构HC板式的价格数据自动更新，也能导致所有指标的重新计算，那样可以暗中输出数据来计算。

## 参数“图表中的最大字节”

“图表中的最大字节”常量限制可行性图表、指标和MQL5程序中HC板式的字节数，对于所有可行性时间序列和服务器来说是有效的，首先，为了节约电脑资源。

当建立有关该常量一个较大值时，需要记住的是，如果深度价格数据对于小的时间架构来说是可行的，用来存储时间序列和指标缓冲区的内存可以是上百个字节，并且达到RAM对客户端程序的限制（2Gb、MS视窗32字节）。

在客户端重启之后，“图表中最大字节”的改变开始生效，该参量的转变原因并不是递交给服务器额外的数据，也不是形成额外的时序列字节，额外价格数据从服务器上要求，时序列的更新于新的限制有关，以防无数据图表卷轴或者数据被MQL5程序调用。

© 2000-2025, MetaQuotes Ltd.

服务器要求的数据成交量与时间架构的“图表中最大字节”的字节要求数量有关，该参数限制并不严格，多数情况下，可利用的字节数字比当前参量值还多。

### 数据有效性

如果这些数据在MQL5程序中以图表或者使用的形式出现，HCC版式中的数据或者为使用HC版式而准备的数据并不能完全可用。

当MQL5程序访问价格数据或者指示值时，他们的可行性是一个确切的时间值或者有保留地从一个确定的时间点开始，事实上，为了节约资源，必要数据的复制并不全部存储在MT5中，只是给出了终端数据库。

所有时间架构的价格历史都从普通的HCC版本数据建立，然后从服务器更新系统更新，重新计算指标，由于这种数据接入能够关闭，甚至这些数据在之前都能使用。

### 终端数据和服务器数据的同步性

自从MQL5程序可以以符号和时间架构的形式调用数据信息，就存在在终端必要的时序列数据没有建立的可能性，或者必要的价格数据没有与交易服务器同步化，因此很难预测潜在时间。

使用潜在循环计算式并不是最好的解决方法，这一事件的唯一例外是脚本，因为有事件要处理，他们没有交替运算方法的选择。对于此种算法的常规指标，和其他潜在循环动作并没有被推荐使用，因为导致所有指标的计算终止，和其他价格数据操作失败。

对于EA交易和指标来说，使用 相等模式 来处理最好不过，在处理OnTick()或 OnCalculate()事件时，必要时序列的接受数据是失败的，此时需要离开事件处理器，在下一个处理器调用的时候适当接入。

## 脚本添加历史的范例

示例是，从交易服务器中接收脚本处理预选对象的请求，脚本已经为运行预选对象图表做了准备；时间架构不能操作，因为如上所述，价格数据是从交易服务器中以一分钟数据形式大包而来的，当时任何预选时序列都在构造中。

记录下所有数据脚本的操作来当成一个分离函数CheckLoadHistory(symbol, timeframe, start_date):

```
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
  }
```

CheckLoadHistory() 函数是一种能被任何操作调用的通用函数（EA交易、脚本或指标）；此前它需要三种输入参量：符号名称，周期和开始日期来表明你所需的价格历史开始运行。

在调用缺失的历史之前，在函数代码中插入格，首先，应该确保符号名称和周期值是正确的：

```
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
```

确定该符号在市场观测窗口的可见性，如下，当向交易服务器发出请求时，符号的历史就可用，如果在市场观测中没有这样的符号，使用 SymbolSelect() 函数可以添加。

© 2000-2025, MetaQuotes Ltd.

```
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
```

现在我们能够收到可用历史交易的起始时间，也许，起始输入函数值，传递到CheckLoadHistory()，是可行性历史，而后交易服务器的请求就不需要了。为了获得第一时间的符号序列，可用使用 SeriesInfoInteger() 函数的 SERIES_FIRSTDATE 修饰语。

```
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
```

下一项重要的检测就是从函数调用检验程序类型，注释，与指标同时发送时序列的更新请求，该种调用更新是不符合要求的。指标中与符号序列相同的要求数据受历史数据约束，因此，固定发生的概率偏高，检验该项目可以使用MQL5InfoInteger()函数中的 MQL5_PROGRAM_TYPE 修饰语。

```
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
      return(-4);
```

如果所有的检测都成功通过，最后要涉及的就是交易服务器，首先找到起始日期，在HCC版式下的分钟日期就可行。使用功能函数SeriesInfoInteger()中SERIES_TERMINAL_FIRSTDATE 修饰语可以调用该值，也可与start_date 常数值做比较。

```
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- 有建立时间序列的加载数据
      if(first_date>0)
        {
         //--- 强制创建时间序列
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- 检测日期
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
```

如果所有的检测都执行通过，却仍然有 CheckLoadHistory() 函数，意味着有必要要求交易服务器调回丢失价格代码，首先，返回 "图表中最大字节"值，使用TerminalInfoInteger() 函数：

```
  int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
```

需要阻止额外调回数据，在服务器中找回字符历史的第一个日期 (忽略周期)，使用已知函数SeriesInfoInteger() 的修饰语 SERIES_SERVER_FIRSTDATE 。

```
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date
      Sleep(5);
```

该请求是异步操作，函数在回收时调用时间晚了5毫秒，直到first_server_date 变量接收到一个值，或者该执行周期被用户终止 (IsStopped() 在此情形下会返回真值)，在起始日期指明正确值，就从向交易服务器要求价格数据时开始。

```
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date," for ",
symbol," does not match to first series date ",first_date);
```

© 2000-2025, MetaQuotes Ltd.

如果服务器的起始日期是 *first_server_date* 比HCC版式的符号 *first_date* 起始日期早，在分类账户中，类似进入将会被输出。

现在向交易服务器发送一个丢失价格数据的请求，该请求以循环的模式开始填满全部函数。

```
   while(!IsStopped())
     {
      //1.  等候如同HHC一样重新建立时间序列与中间历史记录的同步化
      //2.  接收这个时间序列的当前n个柱数
      //      如果柱大于图表中的最大柱，可以退出，工作结束
      //3.  获得重建时间序列开始日期的初始日期并且与开始日期相比较
      //      如果初始日期低于开始日期，则退出，工作结束
      //4.  从服务器请求历史记录的新部分-从最近有效标有'bars'的柱中启动的100柱
     }
```

使用已知手段可以实行前三种方法。

```
   while(!IsStopped())
     {
      //--- 1.等待直到时间序列重建程序结束
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- 2.请求多少柱
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         //--- 超过图表中所画的柱，退出
         if(bars>=max_bars) return(-2);
         //--- 3.  返回时间序列的当前开始日期
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            // 开始日期早于要求日期，任务完成
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //4.  从服务器请求历史记录的新部分-从最近有效标有'bars'的柱中启动的100柱
     }
```

剩下第四步-索取历史，不能直接查阅服务器，如果在HCC板式中的历史不足，任何Copy-function的启动要求都发往服务器，first_date 变量的第一个起始日期是简明的，也有对请求等级的衡量标准，最快的方法是使用 CopyTime() 函数。

当调用函数从时序列复制数据时，应该标记起始参量（字节的数字，复制价格数据的起始），这些都应该在服务器历史中找到，如果只有100字节，从500个指标中复制300字节是无效的，该请求会被视为错误出现不能执行，例如，交易服务器中没有额外历史记录加载。

这就是我们从起始字节指标复制100字节的原因，它会从交易服务器中顺利下载丢失的历史数据，事实上，下载的往往比要求的100字节多一点，服务器也会发送大一些的历史数据。

```
   int copied=CopyTime(symbol,period,bars,100,times);
```

复制操作之后，需要分析复制元素号码，如果尝试失败，*copied*值等于空值并且计数值fail_cnt增加到1，在100次尝试失败后，函数操作就会停止。

```
int fail_cnt=0;
```

© 2000-2025, MetaQuotes Ltd.

```
...
   int copied=CopyTime(symbol,period,bars,100,times);
   if(copied>0)
     {
      //--- 检测数据
      if(times[0]<=start_date)  return(0);  // 复制值更小，准备完毕
      if(bars+copied>=max_bars) return(-2); // 柱多于图表中所画的柱，准备完毕
      fail_cnt=0;
     }
   else
     {
      //--- 成功的话失败尝试少于100
      fail_cnt++;
      if(fail_cnt>=100) return(-5);
      Sleep(10);
     }
```

因此，不仅当前情况的恰当处理在函数中可以实施，而且还能返回结束代码，在调用CheckLoadHistory()函数后可以直接解决获得额外信息，例如，这种方法：

```
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Unknown symbol ",InpLoadedSymbol);                      break;
      case -2 : Print("More requested bars than can be drawn in the chart"); break;
      case -3 : Print("Execution stopped by user");                           break;
      case -4 : Print("Indicator mustn't load its own data");                 break;
      case -5 : Print("Loading failed");                                      break;
      case  0 : Print("All data loaded");                                     break;
      case  1 : Print("Already available data in timeseries are enough");     break;
      case  2 : Print("Timeseries is built from available terminal data");    break;
      default : Print("Execution result undefined");
     }
```

函数的全部代码能在脚本示例中找到，显示出正确的接入操作与请求结果正在处理。

代码：

```
//+------------------------------------------------------------------+
//|                                                TestLoadHistory.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd.      |
//|                                                https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.02"
#property script_show_inputs
//--- 输入参量
input string          InpLoadedSymbol="NZDUSD";    // 被加载的交易品种
input ENUM_TIMEFRAMES InpLoadedPeriod=PERIOD_H1;   // 被加载的周期
input datetime        InpStartDate=D'2006.01.01'; // 开始日期
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   Print("Start load",InpLoadedSymbol+","+GetPeriodName(InpLoadedPeriod),"from",InpSta
//---
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Unknown symbol ",InpLoadedSymbol);              break;
      case -2 : Print("Requested bars more than max bars in chart"); break;
      case -3 : Print("Program was stopped");                        break;
      case -4 : Print("Indicator shouldn't load its own data");      break;
      case -5 : Print("Load failed");                                break;
      case  0 : Print("Loaded OK");                                  break;
      case  1 : Print("Loaded previously");                          break;
      case  2 : Print("Loaded previously and built");                break;
      default : Print("Unknown result");
     }
//---
   datetime first_date;
   SeriesInfoInteger(InpLoadedSymbol,InpLoadedPeriod,SERIES_FIRSTDATE,first_date);
   int bars=Bars(InpLoadedSymbol,InpLoadedPeriod);
   Print("First date ",first_date," - ",bars," bars");
//---
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
   datetime first_date=0;
   datetime times[100];
//--- 检测交易品种 & 周期
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
//--- 检测市场报价是否选定了交易品种
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
//--- 检测数据是否存在
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
//--- 如果是指标的话不需要加载自己的数据
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
      return(-4);
//--- 第二尝试
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- 有建立时间序列加载的数据
```

© 2000-2025, MetaQuotes Ltd.

```
      if(first_date>0)
        {
         //--- 强制创建时间序列
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- 检测日期
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
//--- 图表中来自程序端选项的最大柱
   int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
//--- 加载交易品种历史记录信息
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date
      Sleep(5);
//--- 为加载确定开始日期
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date," for ",symbol,
            " does not match to first series date ",first_date);
//--- 逐步地加载数据
   int fail_cnt=0;
   while(!IsStopped())
     {
      //--- 等待建立时间序列
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- 要求建成的柱
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         if(bars>=max_bars) return(-2);
         //--- 请求初始日期
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //--- 复制部分强制数据加载
      int copied=CopyTime(symbol,period,bars,100,times);
      if(copied>0)
        {
         //--- 数据检测
         if(times[0]<=start_date)  return(0);
         if(bars+copied>=max_bars) return(-2);
         fail_cnt=0;
        }
      else
        {
         //--- 失败尝试少于100
         fail_cnt++;
```

© 2000-2025, MetaQuotes Ltd.

```
         if(fail_cnt>=100) return(-5);
         Sleep(10);
        }
     }
//--- 停止
   return(-3);
  }
//+------------------------------------------------------------------+
//| 返回该周期的字符串值                                             |
//+------------------------------------------------------------------+
string GetPeriodName(ENUM_TIMEFRAMES period)
  {
   if(period==PERIOD_CURRENT) period=Period();
//---
   switch(period)
     {
      case PERIOD_M1:  return("M1");
      case PERIOD_M2:  return("M2");
      case PERIOD_M3:  return("M3");
      case PERIOD_M4:  return("M4");
      case PERIOD_M5:  return("M5");
      case PERIOD_M6:  return("M6");
      case PERIOD_M10: return("M10");
      case PERIOD_M12: return("M12");
      case PERIOD_M15: return("M15");
      case PERIOD_M20: return("M20");
      case PERIOD_M30: return("M30");
      case PERIOD_H1:  return("H1");
      case PERIOD_H2:  return("H2");
      case PERIOD_H3:  return("H3");
      case PERIOD_H4:  return("H4");
      case PERIOD_H6:  return("H6");
      case PERIOD_H8:  return("H8");
      case PERIOD_H12: return("H12");
      case PERIOD_D1:  return("Daily");
      case PERIOD_W1:  return("Weekly");
      case PERIOD_MN1: return("Monthly");
     }
//---
   return("unknown period");
  }
```

© 2000-2025, MetaQuotes Ltd.

# SeriesInfoInteger

关于历史数据的返回信息，有2个变量可供调用。

直接返回属性值。

```
long  SeriesInfoInteger(
   string                      symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES             timeframe,        // 周期
   ENUM_SERIES_INFO_INTEGER    prop_id,          // 属性标识符
   );
```

依靠函数的成果运行返回真值或错误值。

```
bool  SeriesInfoInteger(
   string                      symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES             timeframe,        // 周期
   ENUM_SERIES_INFO_INTEGER    prop_id,          // 属性ID
   long&                       long_var          // 用于获得信息的变量
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*prop_id*

[in] 要求性质标识符，计算 ENUM_SERIES_INFO_INTEGER 值。

*long_var*

[out] 计算安置后的要求性质变量值。

返回值

第一，返回长型值。

第二，如果指定性质可用并且该值添加到 long_var 常量中，返回true,否则返回false.更多细节参照 error，调用 GetLastError()。

示例：

```
void OnStart()
  {
//---
   Print("Total number of bars for the symbol-period at this moment = ",
         SeriesInfoInteger(Symbol(),Period(),SERIES_BARS_COUNT));

   Print("The first date for the symbol-period at this moment = ",
         (datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_FIRSTDATE));
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("The first date in the history for the symbol-period on the server = ",
         (datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_SERVER_FIRSTDATE));

   Print("Symbol data are synchronized = ",
         (bool)SeriesInfoInteger(Symbol(),Period(),SERIES_SYNCHRONIZED));
  }
```

© 2000-2025, MetaQuotes Ltd.

# Bars

为指定交易品种和周期返回历史计算中的字节数量。有两种变量函数可以调用。

要求所有历史字节

```
int  Bars(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe         // 周期
   );
```

为所选时间间隔要求历史字节

```
int  Bars(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time         // 结束日期和时间
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_time*

[in] 与第一元素相一致的字节时间。

*stop_time*

[in] 与最后一个元素相一致的字节时间。

返回值

如果定义start_time 和 stop_time参量，函数返回指定时间间隔的自己数量，否则返回全部字节数量。

注释

如果指定参量的时序列数据通过Bars()函数调用在终端形成，或者时序列数据在函数调用时不在synchronized交易服务器中，函数返回0值。

当请求指定时间间隔的柱形图数量，只有在开盘时间间隔内的柱形图才被考虑。例如，如果一周当前日是星期六，请求就是W1周期的柱数，start_time=last_tuesday 和 stop_time=last_friday，由于W1时间周期的开盘时间始终是星期日并且在指定间隔内没有一个单独的W1柱形图，所以函数将返回0。

所有历史柱形图数量的示例请求：

© 2000-2025, MetaQuotes Ltd.

```
int bars=Bars(_Symbol,_Period);
if(bars>0)
  {
   Print("Number of bars in the terminal history for the symbol-period at the momer
  }
else  //无有效柱
  {
   //--- 交易品种数据不能和服务器数据同步
   bool synchronized=false;
   //--- 循环计数器
   int attempts=0;
   // 进行5次尝试等候同步进行
   while(attempts<5)
     {
      if(SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED))
        {
         //--- 同步化完成，退出
         synchronized=true;
         break;
        }
      //--- 增加计数器
      attempts++;
      //--- 等候10毫秒直至嵌套反复
      Sleep(10);
     }
   //--- 同步化后退出循环
   if(synchronized)
     {
      Print("Number of bars in the terminal history for the symbol-period at the mo
      Print("The first date in the terminal history for the symbol-period at the mo
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));
      Print("The first date in the history for the symbol on the server = ",
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
     }
   //--- 不发生数据同步
   else
     {
      Print("Failed to get number of bars for ",_Symbol);
     }
  }
```

指定间隔中柱形图数量的示例请求：

```
int n;
datetime date1 = D'2016.09.02 23:55'; // 星期五
datetime date2 = D'2016.09.05 00:00'; // 星期一
datetime date3 = D'2016.09.08 00:00'; // 星期四
//---
n=Bars(_Symbol,PERIOD_H1,D'2016.09.02 02:05',D'2016.09.02 10:55');
Print("Number of bars: ",n); // 输出:"柱形图数:8"，计算中考虑H2柱形图，而非H11柱形图
n=Bars(_Symbol,PERIOD_D1,date1,date2);
Print("Number of bars: ",n); // 输出:"柱形图数:1"，由于单独D1(星期一)柱形图开盘时间在间隔内
n=Bars(_Symbol,PERIOD_W1,date2,date3);
Print("Number of bars: ",n); // 输出:"柱形图数:0"，由于没有一个W1柱形图开盘时间在指定间隔内
```

另见

事件处理函数

© 2000-2025, MetaQuotes Ltd.

# BarsCalculated

返回指定指标计算数组的数量。

```
int  BarsCalculated(
   int       indicator_handle,     // 指标处理
   );
```

参量

*indicator_handle*

[in] 指标处理，通过类似指标函数返回。

返回值

在指标缓冲区返回计算数据的数量或者错误中的-1 (数据不能进行计算)。

注释

当需要在创建后立即获得指标数据时，该函数是很有用的 (指标处理可行)。

示例：

```
void OnStart()
  {
   double Ups[];
//--- 设置整理数组的时间序列
   ArraySetAsSeries(Ups,true);
//--- 为分形指标创建处理程序
   int FractalsHandle=iFractals(NULL,0);
//--- 重置错误代码
   ResetLastError();
//--- 复制指标值
   int i,copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Sleep(50);
      for(i=0;i<100;i++)
        {
         if(BarsCalculated(FractalsHandle)>0)
            break;
         Sleep(50);
        }
      copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
      if(copied<=0)
        {
         Print("Failed to copy upper fractals. Error = ",GetLastError(),
         "i = ",i,"    copied = ",copied);
         return;
        }
       else
         Print("Upper fractals copied",
```

© 2000-2025, MetaQuotes Ltd.

```
         "i = ",i,"    copied = ",copied);
     }
   else Print("Upper fractals copied. ArraySize = ",ArraySize(Ups));
  }
```

© 2000-2025, MetaQuotes Ltd.

# IndicatorCreate

函数返回以 MqlParam 类型参量数组为基础建立的特定技术计时器处理。

```
int  IndicatorCreate(
   string           symbol,                    // 交易品种名称
   ENUM_TIMEFRAMES  period,                    // 时间表
   ENUM_INDICATOR   indicator_type,            // ENUM_INDICATOR列举中的指标类型
   int              parameters_cnt=0,          // 参量号
   const MqlParam&  parameters_array[]=NULL,   // 参量数组
   );
```

参量

*symbol*

[in] 交易品种名称，指标计算数据，NULL 表示当前交易品种。

*period*

[in] 时间表的值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*indicator_type*

[in] 指标类型，可以是 ENUM_INDICATOR 值中的一个。

*parameters_cnt*

[in] 参量的数量在_array[]参量数组中传递，数组元素类型有特殊的结构类型 MqlParam。默认，0-参量不传递。如果指定参量的非零数量，参量 *parameters_array* 是强制的，可以传递超过64个参量。

*parameters_array[]=NULL*

[in] MqlParam 类型数组，元素包括类型和每个技术指标输入参数的类型值。

返回值

返回特定技术指标的处理，错误返回 INVALID_HANDLE。

注释

如果IND_CUSTOM类型指标处理建立，数组中第一元素的 *type*域，输入参量*parameters_array*一定是ENUM_DATATYPE 中的TYPE_STRING值，第一元素类型*string_value*域一定包括自定义指标名称。自定义指标必须在MQL5/Indicators客户端或者子目录中编辑。

需要测试的指标是指自动从iCustom()函数调用，如果相应的参数是通过一个通用的常量字符串集合。对于其它情况下（使用IndicatorCreate() 函数或一个非常量字符串设定指标名称）该属性#property tester_indicator是必需的：

```
#property tester_indicator "indicator_name.ex5"
```

在第一调用形式自定义指标中，当传递输入参量时可以加上数据参量。如果"Apply to"参量没有明确规定，默认计算是以 PRICE_CLOSE 值为基础的。

示例：

```
void OnStart()
  {

```

© 2000-2025, MetaQuotes Ltd.

```
   MqlParam params[];
   int      h_MA,h_MACD;
//--- 创建 iMA("EURUSD",PERIOD_M15,8,0,MODE_EMA,PRICE_CLOSE);
   ArrayResize(params,4);
//--- 设置 ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=8;
//--- 设置 ma_shift
   params[1].type         =TYPE_INT;
   params[1].integer_value=0;
//--- 设置 ma_method
   params[2].type         =TYPE_INT;
   params[2].integer_value=MODE_EMA;
//--- 数组 applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=PRICE_CLOSE;
//--- 创建 MA
   h_MA=IndicatorCreate("EURUSD",PERIOD_M15,IND_MA,4,params);
//--- 创建 iMACD("EURUSD",PERIOD_M15,12,26,9,h_MA);
   ArrayResize(params,4);
//--- 设置快速 ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=12;
//--- 设置慢速 ma_period
   params[1].type         =TYPE_INT;
   params[1].integer_value=26;
//--- 为不同性设置平滑周期
   params[2].type         =TYPE_INT;
   params[2].integer_value=9;
//--- 设置指标处理程序为 applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=h_MA;
//--- 创建基于移动平均数的MACD
   h_MACD=IndicatorCreate("EURUSD",PERIOD_M15,IND_MACD,4,params);
//--- 使用指标
//--- . . .
//--- 发布指标 (首先 h_MACD)
   IndicatorRelease(h_MACD);
   IndicatorRelease(h_MA);
  }
```

© 2000-2025, MetaQuotes Ltd.

# IndicatorParameters

基于指定句柄，返回指标输入参数的数量，以及参数的值和类型。

```
int  IndicatorParameters(
   int               indicator_handle,     // 指标处理
   ENUM_INDICATOR&   indicator_type,       // 接收指标类型的变量
   MqlParam&         parameters[]          // 接收参数的数组
   );
```

参数

*indicator_handle*

[in] 指标处理，您需要知道计算的参数数量。

*indicator_type*

[out] 变量，如果ENUM_INDICATOR类型是指标类型将被写入的类型。

*parameters[]*

[out] 接收MqlParam类型值的动态数组，在这里写入指标参数列表。数组大小通过IndicatorParameters()函数返回。

返回值

指定句柄的指标输入参数数量。如果错误则返回-1。要了解有关错误的更多信息，请访问GetLastError()函数。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {

//--- 图表上的窗口数量 至少有一个窗口始终存在）
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
//--- 检查图表窗口
   for(int w=0;w<windows;w++)
     {
      //--- 该窗口/子窗口的指标数量
      int total=ChartIndicatorsTotal(0,w);
      //--- 从窗口获取所有指标
      for(int i=0;i<total;i++)
        {
         //--- 获得指标的缩略名称
         string name=ChartIndicatorName(0,w,i);
         //--- 获得指标句柄
         int handle=ChartIndicatorGet(0,w,name);
         //--- 添加到日志
         PrintFormat("Window=%d,  indicator #%d,  handle=%d",w,i,handle);
         //---
```

© 2000-2025, MetaQuotes Ltd.

```
         MqlParam parameters[];
         ENUM_INDICATOR indicator_type;
         int params=IndicatorParameters(handle,indicator_type,parameters);
         //--- 信息的标头
         string par_info="Short name "+name+", type "
                         +EnumToString(ENUM_INDICATOR(indicator_type))+"\r\n";
         //---
         for(int p=0;p<params;p++)
           {
            par_info+=StringFormat("parameter %d: type=%s, long_value=%d, double_value
                                   p,
                                   EnumToString((ENUM_DATATYPE)parameters[p].type),
                                   parameters[p].integer_value,
                                   parameters[p].double_value,
                                   parameters[p].string_value
                                   );
           }
         Print(par_info);
        }
      //--- 完成窗口的所有指标
     }
//---
  }
```

另见

ChartIndicatorGet()

© 2000-2025, MetaQuotes Ltd.

# IndicatorRelease

如果不使用，函数删除指标缓冲区并释放指标计算空间。

```
bool  IndicatorRelease(
   int       indicator_handle     // 指标处理
   );
```

返回值

如果成功返回真值，否则返回错误值。

注释

如果不再需要，函数允许删除指标缓冲区来节省内存。处理立即删除，计算块会在一定时间内删除（如果不再调用）。

当在策略测试中工作时，不会执行IndicatorRelease()函数。

示例：

```
//+------------------------------------------------------------------+
//|                                        Test_IndicatorRelease.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 输入参量
input int                MA_Period=15;
input int                MA_shift=0;
input ENUM_MA_METHOD     MA_smooth=MODE_SMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
//--- 存储指标处理程序
int MA_handle=INVALID_HANDLE;
//+------------------------------------------------------------------+
//| EA初始化函数                                                     |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建指标处理程序
   MA_handle=iMA(Symbol(),0,MA_Period,MA_shift,MA_smooth,PRICE_CLOSE);
//--- 删除全局变量
   if(GlobalVariableCheck("MA_value"))
      GlobalVariableDel("MA_value");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA订单号函数                                                     |
```

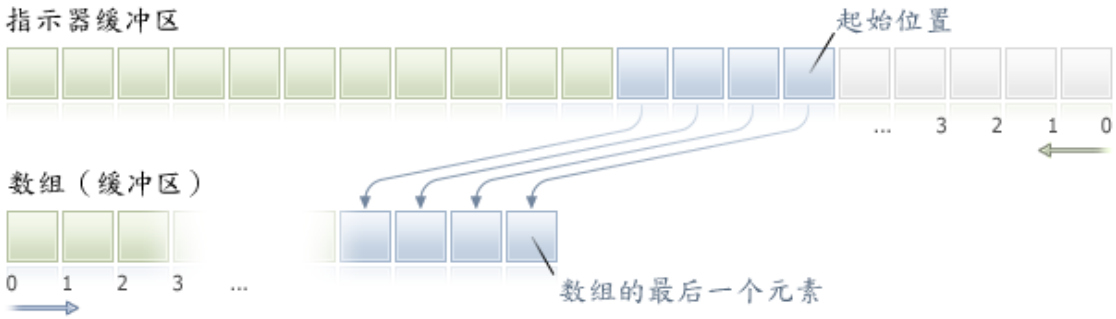
© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 如果全局变量值不存在
   if(!GlobalVariableCheck("MA_value"))
     {
      //--- 在最近2柱获得指标值
      if(MA_handle!=INVALID_HANDLE)
        {
         //--- 指标值的动态数组
         double values[];
         if(CopyBuffer(MA_handle,0,0,2,values)==2 && values[0]!=EMPTY_VALUE)
           {
            //--- 记住倒数第二柱的全局变量值
            if(GlobalVariableSet("MA_value",values[0]))
              {
               //--- 释放指标处理程序
               if(!IndicatorRelease(MA_handle))
                  Print("IndicatorRelease() failed. Error ",GetLastError());
               else MA_handle=INVALID_HANDLE;
              }
            else
               Print("GlobalVariableSet failed. Error ",GetLastError());
           }
        }
     }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

# CopyBuffer

在需要数量中，获得确定指标指定缓冲区的数据。




从一开始执行位置计算复制数据元素（指标缓冲区标引buffer_num）到当前传递位置。例如，起始位置是0代表当前字节（当前字节指标值）。

当复制位置数量的数据时，推荐使用动态数组和buffer类容器缓冲区，因为函数试图分配复制数据的接收数组的大小。如果指标缓冲区（通过使用SetIndexBufer()函数将数组再分配给存储指标值）以buffer类容量数组来使用，就允许局部复制。示例可以在标准客户端包自定义指标Awesome_Oscillator.mq5中找到。

如果需要局部复指标值到另一数组（没有指标缓冲区），应该使用媒介数组，这样可以复制预期数量。如果从中间接收数组，可以管理复制所需元素值的管理之后加到要求的位置。

如果知道所需复制数据的数量，为了阻止过多分配内存，最好操作静态分配缓冲区。

无论目标数组-as_series=true or as_series=false的属性是什么。数据都需要复制，因为最旧的元素放置在数组的记忆内存存储位置，有3种变体可以实现函数调用。

第一位置调用和所需元素数量

```
int  CopyBuffer(
   int       indicator_handle,     // 指标处理
   int       buffer_num,           // 指标缓冲区数
   int       start_pos,            // 启动位置
   int       count,                // 复制总额
   double    buffer[]              // 复制的目标数组
   );
```

开始日期调用和所需元素的数量

```
int  CopyBuffer(
   int       indicator_handle,     // 指标处理
   int       buffer_num,           // 指标缓冲区数
   datetime  start_time,           // 启动日期和时间
   int       count,                // 复制总额
   double    buffer[]              // 复制的目标数组
   );
```

© 2000-2025, MetaQuotes Ltd.

开始调用和要求时间间隔的最后日期

```
int  CopyBuffer(
   int        indicator_handle,     // 指标处理
   int        buffer_num,           // 指标缓冲区数
   datetime   start_time,           // 启动日期和时间
   datetime   stop_time,            // 结束日期和时间
   double     buffer[]              // 复制的目标数组
   );
```

参量

*indicator_handle*

[in] 指标处理，通过类似指标函数返回。

*buffer_num*

[in] 指标缓冲区数量。

*start_pos*

[in] 第一复制元素位置。

*count*

[in] 复制结算数据。

*start_time*

[in] 字节时间，与第一元素相一致。

*stop_time*

[in] 字节时间，与最后元素相一致。

*buffer[]*

[out] 双精度 类型数组。

返回值

返回计算的复制数据或者错误时是-1。

注释

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载 会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

示例：

```
//+------------------------------------------------------------------+
//|                                                TestCopyBuffer3.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图 MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参量
input bool                AsSeries=true;
input int                 period=15;
input ENUM_MA_METHOD      smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE  price=PRICE_CLOSE;
input int                 shift=0;
//--- 指标缓冲区
double                    MABuffer[];
int                       ma_handle;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Parameter AsSeries = ",AsSeries);
   Print("Indicator buffer after SetIndexBuffer() is a timeseries = ",
         ArrayGetAsSeries(MABuffer));
//--- 设置短小的指标名称
   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//--- 设置 AsSeries(依据输入参量)
   ArraySetAsSeries(MABuffer,AsSeries);
   Print("Indicator buffer after ArraySetAsSeries(MABuffer,true); is a timeseries = ",
         ArrayGetAsSeries(MABuffer));
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 检测是否计算了所有数据
   if(BarsCalculated(ma_handle)<rates_total) return(0);
//--- 不复制所有数据
   int to_copy;
   if(prev_calculated>rates_total || prev_calculated<=0) to_copy=rates_total;
   else
     {
      to_copy=rates_total-prev_calculated;
      //--- 常常复制最后的值
      to_copy++;
     }
//--- 复制
   if(CopyBuffer(ma_handle,0,0,to_copy,MABuffer)<=0) return(0);
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

以上示例表明在相同交易品种周期中，指标缓冲区填满另一指标缓冲区的值。

请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

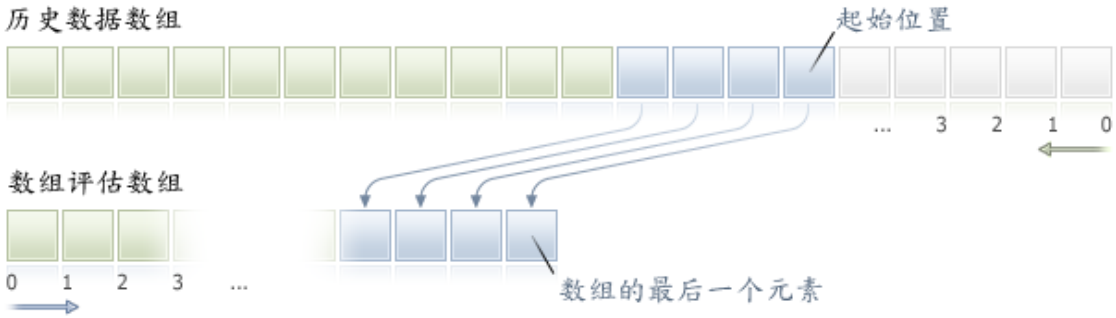
另见

自定义指标属性，SetIndexBuffer

© 2000-2025, MetaQuotes Ltd.

# CopyRates

获得指定交易品种周期的特定MqlRates结构到rates_array数组中。元素复制数据的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区。

无论目标数组- as_series=true or as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyRates(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制数据计算
   MqlRates         rates_array[]     // 复制目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyRates(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 开始日期和时间
   int              count,            // 复制数据计算
   MqlRates         rates_array[]     // 复制目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyRates(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime          start_time,          // 开始日期和时间
   datetime          stop_time,           // 结束日期和时间
   MqlRates          rates_array[]        // 复制目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_time*

[in] 复制的第一元素字节时间。

*start_pos*

[in] 复制的第一元素起始位置。

*count*

[in] 复制的数据点。

*stop_time*

[in] 字节时间，与最后一个复制元素相一致。

*rates_array[]*

[out] MqlRates 类型数组

返回值

返回复制数据数量或者错误时是-1。

注释

如果要求数据的全部间隔不在服务器的可用数据内，函数返回-1。如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载 会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

© 2000-2025, MetaQuotes Ltd.

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和 *stop_time=Last_Friday* ，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

示例：

```
void OnStart()
  {
//---
   MqlRates rates[];
   ArraySetAsSeries(rates,true);
   int copied=CopyRates(Symbol(),0,0,100,rates);
   if(copied>0)
     {
      Print("Bars copied: "+copied);
      string format="open = %G, high = %G, low = %G, close = %G, volume = %d";
      string out;
      int size=fmin(copied,10);
      for(int i=0;i<size;i++)
        {
         out=i+":"+TimeToString(rates[i].time);
         out=out+" "+StringFormat(format,
                                  rates[i].open,
                                  rates[i].high,
                                  rates[i].low,
                                  rates[i].close,
                                  rates[i].tick_volume);
         Print(out);
        }
     }
   else Print("Failed to get history data for the symbol ",Symbol());
  }
```

请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

另见

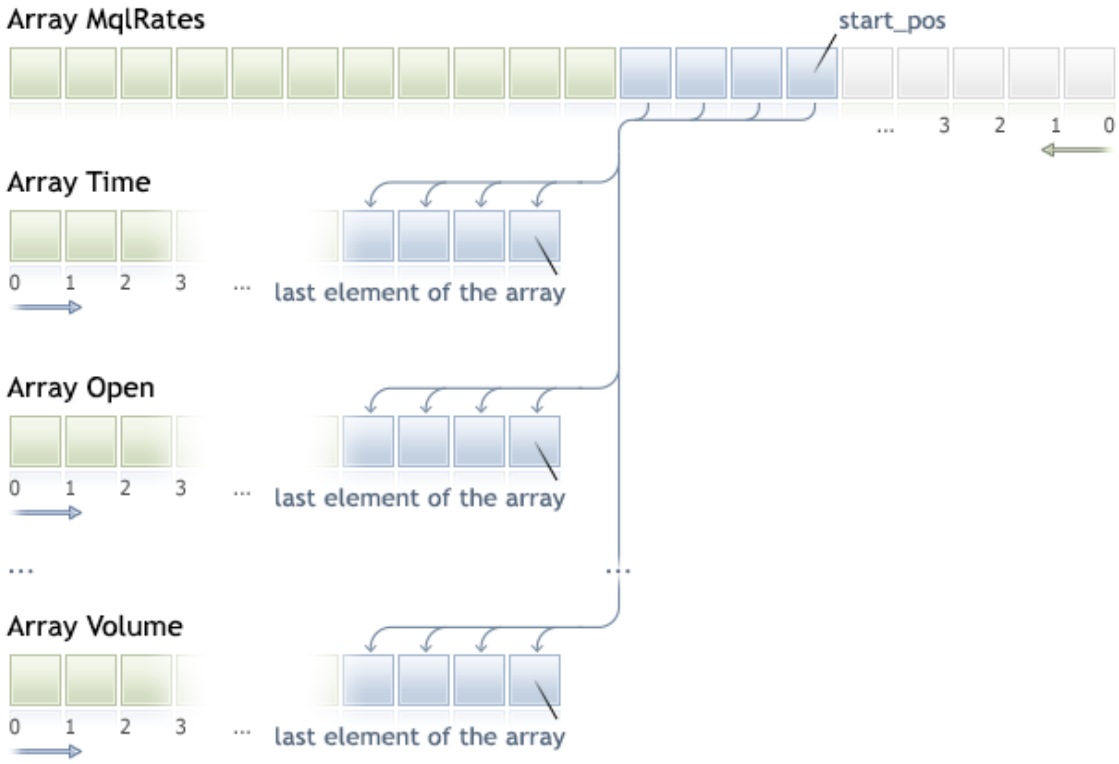
© 2000-2025, MetaQuotes Ltd.

结构和类，时间到字符串，字符串格式

© 2000-2025, MetaQuotes Ltd.

# CopySeries

从MqlRates结构获取指定交易品种周期和指定数量的同步时间序列。数据被收入到指定的数组集中。元素从现在到过去倒数，这意味着起始位置等于0表示当前柱形图。




如果不知道要复制的数据量，建议使用动态数组来接收数组，因为如果数据量超过一个数组的容量，这可能会导致尝试重新分配数组以适应所有请求的数据。

如果您需要复制预定数量的数据，建议使用静态分配缓冲区以避免不必要的内存重新分配。

接收数组的属性 —as_series=true或as_series=false —将被忽略：在复制期间，最早的时间序列元素将被复制到为数组分配的物理内存的开头。

```
int  CopySeries(
   string           symbol_name,       // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,         // 周期
   int              start_pos,         // 起始位置
   int              count,             // 要复制的数量
   ulong            rates_mask,        // 标识组合，以指定所请求的序列
   void&            array1[],          // 接收时间序列的第一个副本数据的数组
   void&            array2[]           // 接收时间序列的第二个副本数据的数组
   ...
   );
```

参数

*symbol_name*

© 2000-2025, MetaQuotes Ltd.

[in] 交易品种。

*timeframe*

[in] 周期。

*start_pos*

[in] 第一个复制的元素索引。

*count*

[in] 已复制元素的数量。

*rates_mask*

[in] 来自ENUM_COPY_RATES枚举中的标识组合。

*array1, array2,...*

[out] 用于从MqlRates结构接收时间序列的相应类型的数组。传递到函数的数组顺序必须与MqlRates结构中字段的顺序相匹配。

返回值

已复制元素的数量或-1（如果发生错误）。

注意

如果请求数据的整个区间超出服务器上可用的数据，函数返回-1。如果请求数据超出TERMINAL_MAXBARS（图表上最大柱形图数），该函数也返回-1。

从指标请求数据时，如果请求的时间序列尚未构建或应从服务器下载，则该函数会立即返回-1。但是，这将启动数据下载/自身构建。

当从EA交易或脚本请求数据时，如果程序端在本地没有合适的数据，则启动从服务器下载，或者如果可以从本地历史构建数据，但数据尚未准备好，则开始构建必要的时间序列。该函数返回超时到期时准备就绪的数据量，但是历史下载继续，该函数在下一次类似请求期间返回更多数据。

## CopySeries和CopyRates之间的区别

CopySeries函数允许在一次调用期间仅将必要的时间序列获取到不同的指定数组中，同时所有的时间序列数据将被同步。这意味着结果数组中特定索引N处的所有值将属于指定交易品种/时间周期对上的同一柱形图。因此，程序员无需通过柱形图开仓时间来确保所有接收到的时间序列同步。

与将完整时间序列集作为MqlRates数组返回的CopyRates不同，CopySeries函数允许程序员仅将所需的时间序列获取为单独的数组。这可以通过指定标识组合来选择时间序列的类型来完成。传递到函数的数组顺序必须与MqlRates结构中字段的顺序相匹配：

```
struct MqlRates
  {
   datetime time;          // 期初
   double   open;          // 开仓价
   double   high;          // 该时段的最高价
   double   low;           // 该时段的最低价
   double   close;         // 平仓价
```

© 2000-2025, MetaQuotes Ltd.

```
   long     tick_volume;  // 报价量
   int      spread;       // 点差
   long     real_volume;  // 交易量
  }
```

因此，如果您需要获取当前交易品种/时间周期的最近100柱形图的*time*, *close*和*real_volume*时间序列的值，您应该使用以下调用：

```
datetime time[];
double   close[];
long     volume[];
CopySeries(NULL,0,0,100,COPY_RATES_TIME|COPY_RATES_CLOSE|COPY_RATES_VOLUME_REAL,time,c
```

注意数组*"time、close、 volume"*顺序 — 必须与 MqlRates结构中字段的顺序相匹配。忽略*rates_mask*中值的顺序。掩码可以如下所示：

```
COPY_RATES_VOLUME_REAL|COPY_RATES_TIME|COPY_RATES_CLOSE
```

例如：

```
//--- 输入参数
input datetime InpDateFrom=D'2022.01.01 00:00:00';
input datetime InpDateTo  =D'2023.01.01 00:00:00';
input uint     InpCount   =20;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 从MqlRates价格结构获取时间序列的数组
   double   open[];
   double   close[];
   float    closef[];
   datetime time1[], time2[];
//--- 将平仓价请求到double数组
   ResetLastError();
   int res1=CopySeries(NULL, PERIOD_CURRENT, 0, InpCount,
                       COPY_RATES_TIME|COPY_RATES_CLOSE, time1, close);
   PrintFormat("1. CopySeries  returns %d values. Error code=%d", res1, GetLastError()
   ArrayPrint(close);

//--- 现在还请求开仓价;为平仓价使用float数组
   ResetLastError();
   int res2=CopySeries(NULL, PERIOD_CURRENT, 0, InpCount,
                       COPY_RATES_TIME|COPY_RATES_CLOSE|COPY_RATES_OPEN, time2, open,
   PrintFormat("2. CopySeries  returns %d values. Error code=%d", res2, GetLastError()
   ArrayPrint(closef);
//--- 对比接收的数据
   if((res1==res2) && (time1[0]==time2[0]))
```

© 2000-2025, MetaQuotes Ltd.

```
     {
      Print("   | Time              |    Open       | Close double | Close float |");
      for(int i=0; i<10; i++)
        {
         PrintFormat("%d | %s |   %.5f    |   %.5f    |   %.5f   |",
                     i, TimeToString(time1[i]), open[i], close[i], closef[i]);
        }
     }
//--- 结果
      1. CopySeries  返回20个值Error code=0
      [ 0] 1.06722 1.06733 1.06653 1.06520 1.06573 1.06649 1.06694 1.06675 1.06684 1.(
      [10] 1.06514 1.06557 1.06456 1.06481 1.06414 1.06394 1.06364 1.06386 1.06239 1.(
      2. CopySeries  返回20个值Error code=0
      [ 0] 1.06722 1.06733 1.06653 1.06520 1.06573 1.06649 1.06694 1.06675 1.06684 1.(
      [10] 1.06514 1.06557 1.06456 1.06481 1.06414 1.06394 1.06364 1.06386 1.06239 1.(
        | Time              |    Open       | Close double | Close float |
      0 | 2023.03.01 17:00 |   1.06660    |   1.06722    |   1.06722   |
      1 | 2023.03.01 18:00 |   1.06722    |   1.06733    |   1.06733   |
      2 | 2023.03.01 19:00 |   1.06734    |   1.06653    |   1.06653   |
      3 | 2023.03.01 20:00 |   1.06654    |   1.06520    |   1.06520   |
      4 | 2023.03.01 21:00 |   1.06520    |   1.06573    |   1.06573   |
      5 | 2023.03.01 22:00 |   1.06572    |   1.06649    |   1.06649   |
      6 | 2023.03.01 23:00 |   1.06649    |   1.06694    |   1.06694   |
      7 | 2023.03.02 00:00 |   1.06683    |   1.06675    |   1.06675   |
      8 | 2023.03.02 01:00 |   1.06675    |   1.06684    |   1.06684   |
      9 | 2023.03.02 02:00 |   1.06687    |   1.06604    |   1.06604   |
//---
  }
```

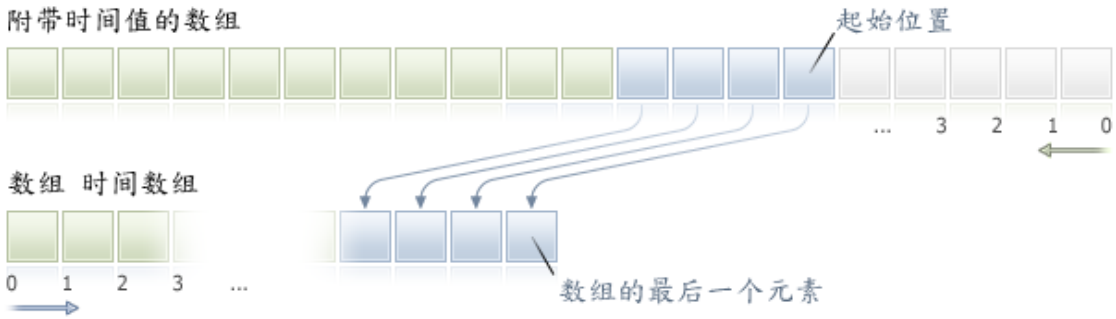
另见

结构和类, CopyRates

© 2000-2025, MetaQuotes Ltd.

# CopyTime

函数在特定量中获得指定交易品种周期的特定开盘时间数据到time_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyTime(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   datetime         time_array[]      // 复制开仓时间的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyTime(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   datetime         time_array[]      // 复制开仓时间的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyTime(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime          time_array[]      // 复制开仓时间的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制第一元素的起始位置。

*count*

[in] 复制的数据点。

*start_time*

[in] 复制的第一元素起始时间。

*stop_time*

[in] 复制最后元素相一致的字节时间。

*time_array[]*

[out] 日期时间 类型数组

返回值

返回复制数据点或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS 图表中的最大字节量）是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于（早于）或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值（交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间）始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值（成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间）是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表 *start_time=Last_Tuesday* 和 *stop_time=Last_Friday* ，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

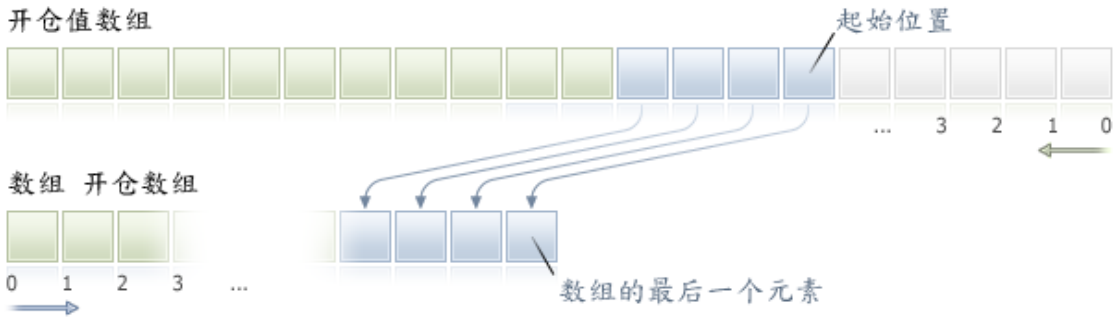
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyOpen

函数在特定量中获得指定交易品种周期的特定开盘价格数据到open_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用 动态数组 为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyOpen(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   double           open_array[]      // 复制开仓时间的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyOpen(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   double           open_array[]      // 用于开盘柱的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyOpen(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   double           open_array[]      // 用于开盘柱的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制的第一元素起始位置。

*count*

[in] 复制的数据点。

*start_time*

[in] 复制的第一元素起始时间。

*stop_time*

[in] 复制的最后元素的起始时间。

*open_array[]*

[out] 双精度 类型数组

返回值

返回数组中元素数量或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS 图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表 *start_time=Last_Tuesday* 和 *stop_time=Last_Friday* ，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的*start_pos*=0和*count*=1第一种形式。

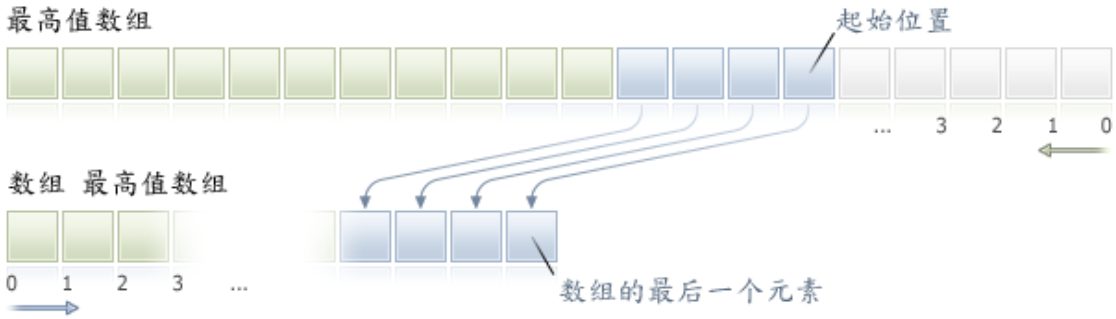
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyHigh

函数在特定量中获得指定交易品种周期的特定最高字节价格数据到high_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyHigh(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   double           high_array[]      // 复制的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyHigh(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   double           high_array[]      // 复制的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyHigh(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   double           high_array[]      // 复制的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制的第一元素起始位置。

*count*

[in] 复制数据点。

*start_time*

[in] 复制第一个元素的起始时间。

*stop_time*

[in] 字节时间，与复制最后元素相一致。

*high_array[]*

[out] 双精度 类型数组。

返回值

返回数组中元素数量或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和*stop_time=Last_Friday*，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

示例：

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An example for output of the High[i] and Low[i]"
#property description "for a random chosen bars"

double High[],Low[];
//+------------------------------------------------------------------+
//| 为指定柱指数获得低价                                             |
//+------------------------------------------------------------------+
double iLow(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double low=0;
   ArraySetAsSeries(Low,true);
   int copied=CopyLow(symbol,timeframe,0,Bars(symbol,timeframe),Low);
   if(copied>0 && index<copied) low=Low[index];
   return(low);
  }
//+------------------------------------------------------------------+
//| 为指定柱指数获得高价                                             |
//+------------------------------------------------------------------+
double iHigh(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double high=0;
   ArraySetAsSeries(High,true);
   int copied=CopyHigh(symbol,timeframe,0,Bars(symbol,timeframe),High);
   if(copied>0 && index<copied) high=High[index];
   return(high);
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 每个订单号为带有指数的柱输出高低值，
//--- 等同第二订单号
   datetime t=TimeCurrent();
   int sec=t%60;
   printf("High[%d] = %G  Low[%d] = %G",
          sec,iHigh(Symbol(),0,sec),
          sec,iLow(Symbol(),0,sec));
  }
```

© 2000-2025, MetaQuotes Ltd.

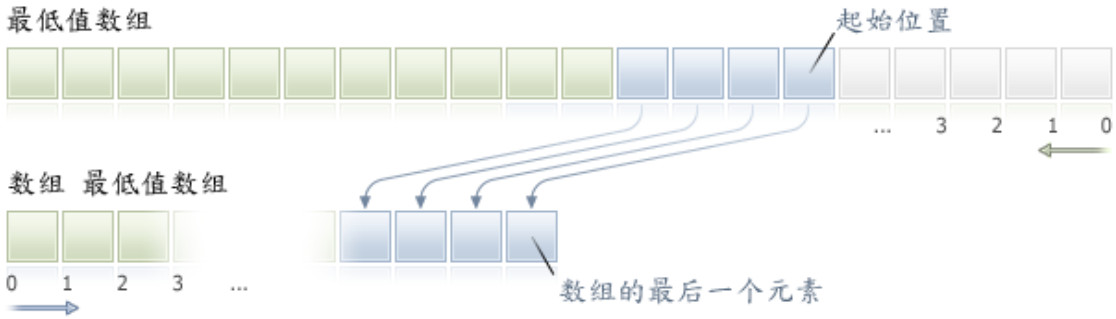
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyLow

函数在特定量中获得指定交易品种周期的特定最低字节价格数据到low_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyLow(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   int               start_pos,        // 启动位置
   int               count,            // 复制的数据计算
   double            low_array[]       // 复制的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyLow(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   datetime          start_time,       // 启动日期和时间
   int               count,            // 复制的数据计算
   double            low_array[]       // 复制的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyLow(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   datetime          start_time,       // 启动日期和时间
   datetime          stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   double           low_array[]      // 复制的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制的第一元素起始位置。

*count*

[in] 复制数据点。

*start_time*

[in] 字节时间，复制第一元素相一致。

*stop_time*

[in] 字节时间，复制的最后元素相一致。

*low_array[]*

[out] 双精度 类型数组。

返回值

返回数组中元素数量或者 错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和*stop_time=Last_Friday*，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

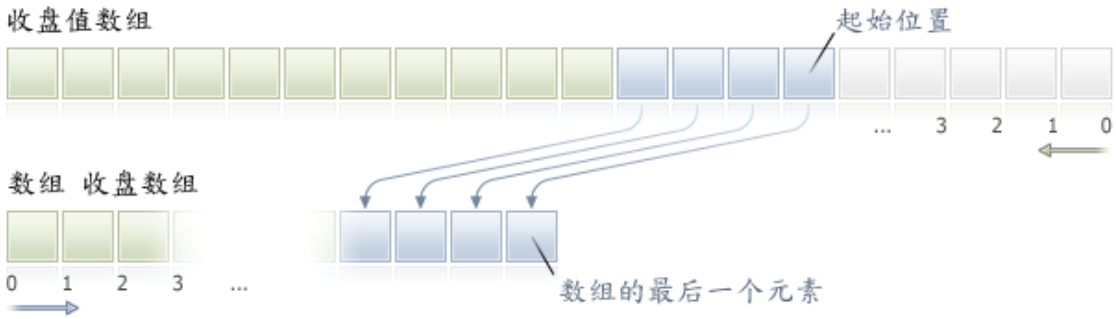
另见

复制最高价

© 2000-2025, MetaQuotes Ltd.

# CopyClose

函数在特定量中获得指定交易品种周期的特定收盘价格数据到close_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyClose(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   double           close_array[]     // 复制的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyClose(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   double           close_array[]     // 复制的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyClose(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   double           close_array[]      // 复制的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制第一元素起始位置。

*count*

[in] 复制日期点。

*start_time*

[in] 复制第一元素起始位置。

*stop_time*

[in] 字节时间，复制最后元素相一致。

*close_array[]*

[out] 双精度 类型数组。

返回值

返回数组中元素数量或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和*stop_time=Last_Friday*，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

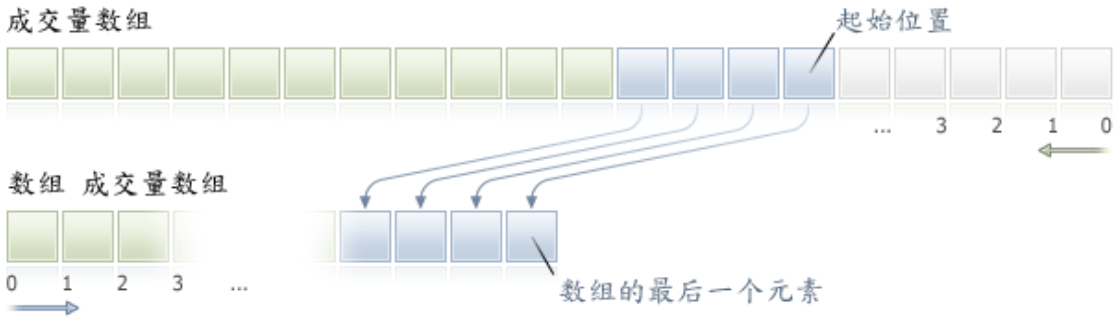
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyTickVolume

函数在特定量中获得指定交易品种周期的特定点击成交量数据到volum e_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyTickVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   long             volume_array[]    // 订单号交易量的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyTickVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   long             volume_array[]    // 订单号交易量的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyTickVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 开始日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   long              volume_array[]    // 订单号交易量的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制的第一元素开始位置。

*count*

[in] 复制数据点。

*start_time*

[in] 复制第一元素起始时间。

*stop_time*

[in] 字节时间，复制最后元素相一致。

*volume_array[]*

[out] 长整型类型数组

返回值

返回数组中元素数量或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和*stop_time=Last_Friday*，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

示例：

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 订单号交易量图
#property indicator_label1  "TickVolume"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  C'143,188,139'
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参量
input int      bars=3000;
//--- 指标缓冲区
double         TickVolumeBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,TickVolumeBuffer,INDICATOR_DATA);
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      long timeseries[];
      ArraySetAsSeries(timeseries,true);
      int prices=CopyTickVolume(Symbol(),0,0,bars,timeseries);
      for(int i=0;i<rates_total-prices;i++) TickVolumeBuffer[i]=0.0;
```

© 2000-2025, MetaQuotes Ltd.

```
      for(int i=0;i<prices;i++) TickVolumeBuffer[rates_total-1-i]=timeseries[prices-1-
      Print("We have received the following number of TickVolume values: "+prices);
     }
   else
     {
      long timeseries[];
      int prices=CopyTickVolume(Symbol(),0,0,1,timeseries);
      TickVolumeBuffer[rates_total-1]=timeseries[0];
     }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

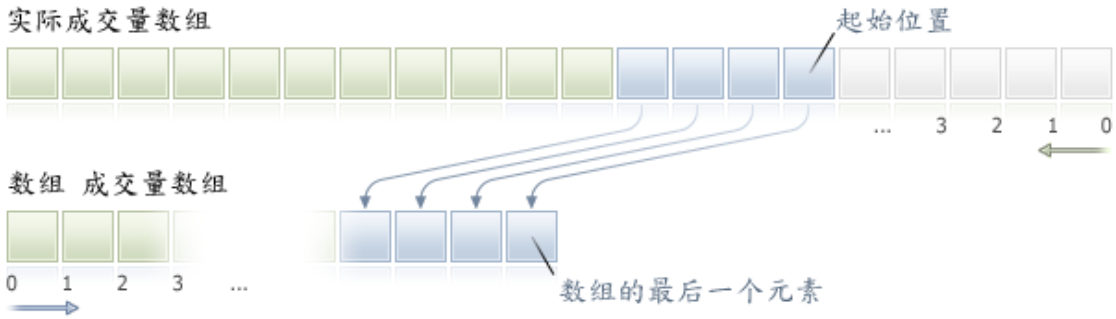
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyRealVolumn

函数在特定量中获得指定交易品种周期的特定交易成交量数据到volume_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用 静态分配缓冲区 。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopyRealVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   int              start_pos,        // 启动位置
   int              count,            // 复制的数据计算
   long             volume_array[]    // 交易量值的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopyRealVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   int              count,            // 复制的数据计算
   long             volume_array[]    // 交易量值的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopyRealVolume(
   string           symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES  timeframe,        // 周期
   datetime         start_time,       // 启动日期和时间
   datetime         stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   long              volume_array[]    // 交易量值的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制第一元素起始位置。

*count*

[in] 复制数据点。

*start_time*

[in] 复制第一元素起始时间。

*stop_time*

[in] 字节时间，复制最后元素相一致。

*volume_array[]*

[out] 长整型类型数组

返回值

返回数组中元素数量或者错误情况下-1 。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS (图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表*start_time=Last_Tuesday* 和*stop_time=Last_Friday*，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

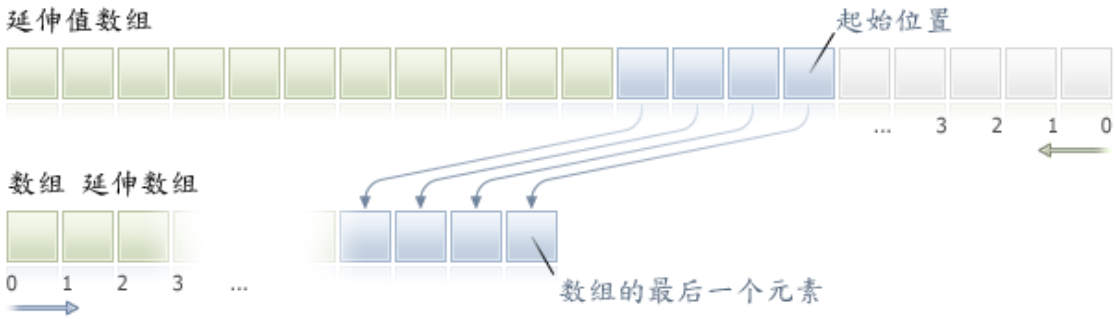
请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopySpread

函数在特定量中获得指定交易品种周期的特定点差价值数据到spread_array数据中。元素注释的命令是从现在到过去，例如，从当前位置0开始表示当前字节。




当复制未知数量数据时，推荐使用动态数组为目标数组，因为如果要求数据账户少于（或多于）目标数值的长度，函数试图重新分配内存以完全适应要求数据。

如果知道所需复制的数据数量，为了阻止内存过多分配，最好使用静态分配缓冲区。

无论目标数组-as_series=true 或者 as_series=false的属性是什么，数据就会复制，以至于最旧的数组元素会在记忆内存开始中放置。有3个变量函数可供调用。

调用第一位置和要求元素的数量

```
int  CopySpread(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   int               start_pos,        // 启动位置
   int               count,            // 复制的数据计算
   int               spread_array[]    // 点差值的目标数组
   );
```

调用起始位置日期和要求元素数量

```
int  CopySpread(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   datetime          start_time,       // 启动日期和时间
   int               count,            // 复制的数据计算
   int               spread_array[]    // 点差值的目标数组
   );
```

调用起始位置和要求时间间隔的日期末尾

```
int  CopySpread(
   string            symbol_name,      // 交易品种名称
   ENUM_TIMEFRAMES   timeframe,        // 周期
   datetime          start_time,       // 启动日期和时间
   datetime          stop_time,        // 结束日期和时间
```

© 2000-2025, MetaQuotes Ltd.

```
   int               spread_array[]    // 点差值的目标数组
   );
```

参量

*symbol_name*

[in] 交易品种名称。

*timeframe*

[in] 周期。

*start_pos*

[in] 复制第一元素起始位置。

*count*

[in] 复制数据点。

*start_time*

[in] 复制第一元素起始位置。

*stop_time*

[in] 字节时间，复制最后元素相一致。

*spread_array[]*

[out] 整型 类型数组

返回值

返回数组中元素数量或者错误情况下-1。

注释

如果要求的数据的全部间隔没有在可行数据的服务器中，函数返回-1，如果外部数据 TERMINAL_MAXBARS 图表中的最大字节量)是要求的，函数也会返回-1。

当向指标要求数据，如果要求的时序列没有建立或者需要从服务器上下载，函数很快会返回-1，但是下载/建立的过程将会开始。

当EA交易或脚本要求数据时，来自服务器的下载会开始，如果终端本地没有这些数据，要求建立的时序列就会开始，如果数据可以从本地历史建立但尚未准备好。函数在超时期满时，会返回一定数量的准备好的数据，但是历史下载会继续，在下一个类似要求函数中会返回更多数据。

当通过起始日期和所需的元素数量请求数据时，只有日期小于 (早于)或等于指定日期的数据。这意味着，任何柱的开盘时间的返回值 (交易量，点差，指标缓冲区，开盘价，最高价，最低价，收盘价或开盘时间)始终少于或等于指定的值。

当在日期指定范围内要求数据，只有间隔中的数据才能返回，间隔建立并指到秒。这表示，任意字节的开始时间，返回的值 (成交量，传播，指标缓冲区的值，开仓价，最高阶，最低价，收盘价或者开仓时间)是间隔要求范围内的。

因此，如果当前日期是星期六，想要复制指定的一周的时间表 *start_time=Last_Tuesday* 和 *stop_time=Last_Friday* ，函数的返回值是0，因为每周开盘时间是星期日，但是一周字节不能分成特殊的间隔。

© 2000-2025, MetaQuotes Ltd.

如果需要返回与当前未完成字节相类似的值，可以调用指定的 *start_pos*=0 和 *count*=1第一种形式。

示例：

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 点差图
#property indicator_label1  "Spread"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参量
input int      bars=3000;
//--- 指标缓冲区
double         SpreadBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,SpreadBuffer,INDICATOR_DATA);
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      int spread_int[];
      ArraySetAsSeries(spread_int,true);
      int spreads=CopySpread(Symbol(),0,0,bars,spread_int);
      Print("We have received the following number of Spread values: ",spreads);
```

© 2000-2025, MetaQuotes Ltd.

```
      for (int i=0;i<spreads;i++)
      {
      SpreadBuffer[rates_total-1-i]=spread_int[i];
      if(i<=30) Print("spread["+i+"] = ",spread_int[i]);
      }
     }
   else
     {
      double Ask,Bid;
      Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
      Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
      Comment("Ask = ",Ask,"  Bid = ",Bid);
      SpreadBuffer[rates_total-1]=(Ask-Bid)/Point();
     }
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
```

请见对象绑定方法部分请求数据历史记录的详细示例。该部分可用的脚本显示了如何获得最近1000柱上指标iFractals 的值和如何显示过去10的上下分形图。类似的技术可以用于所有丢失数据的指标并通常使用以下类型绘制：

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

© 2000-2025, MetaQuotes Ltd.

# CopyTicks

该函数以 MqlTick 格式接收报价到ticks_array。 在这种情况下，从过去到现在索引报价，例如0号报价是数组中最早的报价。对于报价分析，请检查*flags* 字段，它会显示报价中实际变化的内容。

```
int  CopyTicks(
   string           symbol_name,           // 交易品种名称
   MqlTick&         ticks_array[],         // 报价接收数组
   uint             flags=COPY_TICKS_ALL,  // 决定接收报价类型的标记
   ulong            from=0,                // 您想请求报价的日期
   uint             count=0                // 您想接收报价的数量
   );
```

参数

*symbol_name*

[in] 交易品种。

*ticks_array*

[out] 接收报价MqlTick 类型的数组。

*flags*

[in] 确定请求报价类型的标记。COPY_TICKS_INFO - 卖价和/或买价变化的报价，COPY_TICKS_TRADE -最后价和交易量变化的报价， COPY_TICKS_ALL -全部报价。对于任何请求类型，之前的报价值添加到其余的MqlTick结构的字段。

*from*

[in] 您想请求报价的开始日期。从1970.01.01开始以毫秒计算。如果*from*=0，将返回最近的*count* 报价。

*count*

[in] 请求报价的数量。如果没有指定'from' 和 'count' 参数，所有可用的现有报价 (不超过2000)将被写入ticks_array[]。

返回值

复制报价的数量或错误情况下的-1。

进一步注释

CopyTicks() 函数允许请求和分析所有接收的报价。第一次调用CopyTicks() 即开始同步硬盘上存储的交易品种报价数据库。如果本地数据库没有提供所有请求的报价，那么丢失的报价将会自动从交易服务器下载。从CopyTicks()指定的*from* 开始日期直到当前时刻的报价将会同步。在那之后，到达该交易品种的所有报价将被添加到报价数据库，因此保持同步状态。

如果没有指定*from* 和 *count* 参数，所有可用的现有报价 (不超过2000)将被写入*ticks_array[]*。*flags* 参数允许指定所需报价类型。

COPY_TICKS_INFO -返回卖价和/或买价变化的报价。其他字段的数据也将被添加。例如，如果只有卖价改变，*买价* 和 *交易量* 字段将填写最近得知的值。若要找出实际变化的内容，请分析具有TICK_FLAG_BID和/或TICK_FLAG_ASK值的*flags*字段。如果报价有零值的卖价和买价，那么标记会显示这些数据已经更改(*flags*=TICK_FLAG_BID|TICK_FLAG_ASK)，这意味着订单簿 (市场深度)为空。换句话说，没有买入和卖出订单。

© 2000-2025, MetaQuotes Ltd.

COPY_TICKS_TRADE - 返回最后价和交易量变化的报价。其他字段的数据也将被添加，例如最近得到的卖价和买价值将在适当字段指明。若要找出实际变化的内容，请分析具有TICK_FLAG_LAST 和 TICK_FLAG_VOLUME值的*flags* 字段。

COPY_TICKS_ALL -返回任何变化的所有报价。未改变的字段将会填写最近得到的值。

调用COPY_TICKS_ALL标记的CopyTicks() 立即返回请求间隔的全部报价，而其他模式的调用则需要一定的时间来处理和选择报价，因此它们不提供显著的速度优势。

请求报价时 (既有 COPY_TICKS_INFO 还有 COPY_TICKS_TRADE)，每个报价都包括直至报价时间为止的完整的价格信息(*卖价，买价，最后价*和 *交易量*)。该功能主要用于更轻松的分析每次报价时的交易状态，所以不需要请求深度报价历史和搜索其他字段的值。

**在指标中，CopyTicks()函数返回结果：** 从指标调用时，CopyTick() 立即返回交易品种的所有可用报价，并且如果可用数据不足时，将启动同步报价数据库。一个交易品种的所有指标在一个常见线程中运行，所有指标不会等候同步完成。同步之后，CopyTicks() 将返回下次调用期间所有的请求报价。在指标中，每次报价到达之后都会调用OnCalculate() 函数。

**CopyTicks() 可以等候45秒来等待EA交易和脚本的结果：** 不同于指标，每个EA交易和脚本都在独立的线程中运行，因此可以等待45秒，直到同步完成。如果所需的报价数在此期间同步失败，那么CopyTicks() 将通过timeout返回可用报价并将继续同步。EA交易中的OnTick() 并不是每个报价处理程序，它仅通知EA交易有关市场中的变化。如果产生大量变化：程序端可以同时做出几个报价，但是OnTick()将只能调用一次来通知EA最新市场状态。

**数据返回速度：** 每个工具，程序端可以存储快速访问缓存4096个最新报价(运行市场深度，是65536个报价用于交易品种。如果当前交易期请求的报价超出了缓存，CopyTicks()则会调用存储在程序端内存的报价。这些请求需要更多的执行时间。最慢的请求就是那些其他日期的请求报价，因为在这种情况下从磁盘读取数据。

例如：

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- Requesting 100 million ticks to be sure we receive the entire tick history
input int      getticks=100000000; // The number of required ticks
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int     attempts=0;     // Count of attempts
   bool    success=false;  // The flag of a successful copying of ticks
   MqlTick tick_array[];   // Tick receiving array
   MqlTick lasttick;       // To receive last tick data
   SymbolInfoTick(_Symbol,lasttick);
//--- Make 3 attempts to receive ticks
   while(attempts<3)
     {
      //--- Measuring start time before receiving the ticks
      uint start=GetTickCount();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- Requesting the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
      int received=CopyTicks(_Symbol,tick_array,COPY_TICKS_ALL,1,getticks);
      if(received!=-1)
        {
         //--- Showing information about the number of ticks and spent time
         PrintFormat("%s: received %d ticks in %d ms",_Symbol,received,GetTickCount()-
         //--- If the tick history is synchronized, the error code is equal to zero
         if(GetLastError()==0)
           {
            success=true;
            break;
           }
         else
            PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d
            _Symbol,received,GetTickCount()-start,_LastError);
        }
      //--- Counting attempts
      attempts++;
      //--- A one-second pause to wait for the end of synchronization of the tick data
      Sleep(1000);
     }
//--- Receiving the requested ticks from the beginning of the tick history failed in t
   if(!success)
     {
      PrintFormat("Error! Failed to receive %d ticks of %s in three attempts",getticks
      return;
     }
   int ticks=ArraySize(tick_array);
//--- Showing the time of the first tick in the array
   datetime firstticktime=tick_array[ticks-1].time;
   PrintFormat("Last tick time = %s.%03I64u",
               TimeToString(firstticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_ar
//--- выведем время последнего тика в массиве
   datetime lastticktime=tick_array[0].time;
   PrintFormat("First tick time = %s.%03I64u",
               TimeToString(lastticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_ar

//---
   MqlDateTime today;
   datetime current_time=TimeCurrent();
   TimeToStruct(current_time,today);
   PrintFormat("current_time=%s",TimeToString(current_time));
   today.hour=0;
   today.min=0;
   today.sec=0;
   datetime startday=StructToTime(today);
   datetime endday=startday+24*60*60;
   if((ticks=CopyTicksRange(_Symbol,tick_array,COPY_TICKS_ALL,startday*1000,endday*100
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("CopyTicksRange(%s,tick_array,COPY_TICKS_ALL,%s,%s) failed, error %d
                  _Symbol,TimeToString(startday),TimeToString(endday),GetLastError());
      return;
     }
   ticks=MathMax(100,ticks);
//--- Showing the first 100 ticks of the last day
   int counter=0;
   for(int i=0;i<ticks;i++)
     {
      datetime time=tick_array[i].time;
      if((time>=startday) && (time<endday) && counter<100)
        {
         counter++;
         PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
        }
     }
//--- Showing the first 100 deals of the last day
   counter=0;
   for(int i=0;i<ticks;i++)
     {
      datetime time=tick_array[i].time;
      if((time>=startday) && (time<endday) && counter<100)
        {
         if(((tick_array[i].flags&TICK_FLAG_BUY)==TICK_FLAG_BUY) || ((tick_array[i].fl
           {
            counter++;
            PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
           }
        }
     }
  }
//+------------------------------------------------------------------+
//| Returns the string description of a tick                         |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03d ",
                            TimeToString(tick.time),tick.time_msc%1000);
//--- Checking flags
   bool buy_tick=((tick.flags&TICK_FLAG_BUY)==TICK_FLAG_BUY);
   bool sell_tick=((tick.flags&TICK_FLAG_SELL)==TICK_FLAG_SELL);
   bool ask_tick=((tick.flags&TICK_FLAG_ASK)==TICK_FLAG_ASK);
   bool bid_tick=((tick.flags&TICK_FLAG_BID)==TICK_FLAG_BID);
   bool last_tick=((tick.flags&TICK_FLAG_LAST)==TICK_FLAG_LAST);
   bool volume_tick=((tick.flags&TICK_FLAG_VOLUME)==TICK_FLAG_VOLUME);
//--- Checking trading flags in a tick first
   if(buy_tick || sell_tick)
     {
      //--- Forming an output for the trading tick
```

© 2000-2025, MetaQuotes Ltd.

```
      desc=desc+(buy_tick?StringFormat("Buy Tick: Last=%G Volume=%d ",tick.last,tick.v
      desc=desc+(sell_tick?StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last,tick
      desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
      desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
      desc=desc+"(Trade tick)";
     }
   else
     {
      //--- Form a different output for an info tick
      desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
      desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
      desc=desc+(last_tick?StringFormat("Last=%G ",tick.last):"");
      desc=desc+(volume_tick?StringFormat("Volume=%d ",tick.volume):"");
      desc=desc+"(Info tick)";
     }
//--- Returning tick description
   return desc;
  }
//+------------------------------------------------------------------+
/* Example of the output
Si-12.16: received 11048387 ticks in 4937 ms
Last tick time = 2016.09.26 18:32:59.775
First tick time = 2015.06.18 09:45:01.000
1.  2016.09.26 09:45.249 Ask=65370 Bid=65370 (Info tick)
2.  2016.09.26 09:47.420 Ask=65370 Bid=65370 (Info tick)
3.  2016.09.26 09:50.893 Ask=65370 Bid=65370 (Info tick)
4.  2016.09.26 09:51.827 Ask=65370 Bid=65370 (Info tick)
5.  2016.09.26 09:53.810 Ask=65370 Bid=65370 (Info tick)
6.  2016.09.26 09:54.491 Ask=65370 Bid=65370 (Info tick)
7.  2016.09.26 09:55.913 Ask=65370 Bid=65370 (Info tick)
8.  2016.09.26 09:59.350 Ask=65370 Bid=65370 (Info tick)
9.  2016.09.26 09:59.678 Bid=65370 (Info tick)
10. 2016.09.26 10:00.000 Sell Tick: Last=65367 Volume=3 (Trade tick)
11. 2016.09.26 10:00.000 Sell Tick: Last=65335 Volume=45 (Trade tick)
12. 2016.09.26 10:00.000 Sell Tick: Last=65334 Volume=95 (Trade tick)
13. 2016.09.26 10:00.191 Sell Tick: Last=65319 Volume=1 (Trade tick)
14. 2016.09.26 10:00.191 Sell Tick: Last=65317 Volume=1 (Trade tick)
15. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=1 (Trade tick)
16. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=10 (Trade tick)
17. 2016.09.26 10:00.191 Sell Tick: Last=65315 Volume=5 (Trade tick)
18. 2016.09.26 10:00.191 Sell Tick: Last=65313 Volume=3 (Trade tick)
19. 2016.09.26 10:00.191 Sell Tick: Last=65307 Volume=25 (Trade tick)
20. 2016.09.26 10:00.191 Sell Tick: Last=65304 Volume=1 (Trade tick)
21. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=1 (Trade tick)
22. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=10 (Trade tick)
23. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=5 (Trade tick)
24. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=1 (Trade tick)
25. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=6 (Trade tick)
26. 2016.09.26 10:00.191 Sell Tick: Last=65299 Volume=1 (Trade tick)
```

© 2000-2025, MetaQuotes Ltd.

```
27. 2016.09.26 10:00.191 Bid=65370 (Info tick)
28. 2016.09.26 10:00.232 Ask=65297 (Info tick)
29. 2016.09.26 10:00.276 Sell Tick: Last=65291 Volume=31 (Trade tick)
30. 2016.09.26 10:00.276 Sell Tick: Last=65290 Volume=1 (Trade tick)
*/
```

另见

SymbolInfoTick, 当前价格的结构, OnTick()

© 2000-2025, MetaQuotes Ltd.

# CopyTicksRange

该函数可以在指定日期范围内以MqlTick格式接收报价到ticks_array。从过去到现在索引报价意味着0号报价是数组中最早的报价。对于报价分析，请检查*flags* 字段，它会显示实际变化的内容。

```
int  CopyTicksRange(
   const string      symbol_name,            //交易品种名称
   MqlTick&          ticks_array[],          // 报价接收数组
   uint              flags=COPY_TICKS_ALL,   // 定义接收报价类型的标记
   ulong             from_msc=0,             // 请求报价开始的日期
   ulong             to_msc=0                // 请求报价截至日期
   );
```

参数

*symbol_name*

[in] 交易品种。

*ticks_array*

[out] MqlTick 接收报价的静态或动态数组。如果静态数组不能保存来自请求间隔的全部报价，则会尽可能多的接收报价量。在这种情况下，该函数生成ERR_HISTORY_SMALL_BUFFER (4407)错误。

*flags*

[in] 确定请求报价类型的标记。COPY_TICKS_INFO - 卖价和/或买价变化的报价，COPY_TICKS_TRADE -最后价和交易量变化的报价， COPY_TICKS_ALL -全部报价。对于任何请求类型，之前的报价值添加到其余的MqlTick结构的字段。

*from_msc*

[in] 您想请求报价的开始日期。从1970.01.01开始以毫秒计算。如果没有指定*from_msc* 参数，则从历史记录中最初的报价开始发送。报价时间 >= from_msc 已发送。

*to_msc*

[in] 您想请求报价的截至日期。从1970.01.01开始，以毫秒计算。报价时间 <= to_msc 已发送。如果没有指定*to_msc* 参数，则发送截至到历史记录结束的报价。

返回值

复制的报价数或错误时显示-1。GetLastError()可能返回以下错误：

- ERR_HISTORY_TIMEOUT -报价同步等待时间到达上限，函数已发送具有的全部报价。
- ERR_HISTORY_SMALL_BUFFER -静态缓冲区过小。只有数组可存储的数量已经发送。
- ERR_NOT_ENOUGH_MEMORY - 接收指定范围的历史记录到动态报价数组的内存不足。无法为报价数组分配足够的内存。

注意

CopyTicksRange()函数用于严格的从一个指定范围请求报价，例如，从历史的某一天。与此同时，CopyTicks() 还可以仅指定开始日期，例如 -接收月初至当前时间的全部报价。

另见

SymbolInfoTick, 当前价格结构, OnTick, CopyTicks

© 2000-2025, MetaQuotes Ltd.

# iBars

返回可用于历史记录的相应交易品种和周期的柱形图的数量。

```
int  iBars(
   const string           symbol,          // 交易品种
   ENUM_TIMEFRAMES        timeframe        // 周期
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

返回值

可用于历史记录的对应交易品种和周期的柱形图的数量，最多不超过平台设置中"图表最大柱形图"可允许的参数。

例如：

```
  Print("Bar count on the 'EURUSD,H1' is ",iBars("EURUSD",PERIOD_H1));
```

另见

Bars

© 2000-2025, MetaQuotes Ltd.

# iBarShift

根据时间搜索柱形图。这个函数返回与指定时间对应的柱形图指数。

```
int  iBarShift(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   datetime             time,            // 时间
   bool                 exact=false      // 模式
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。PERIOD_CURRENT 表示当前图表周期。

*时间*

[in] 要搜索的时间值。

*exact=false*

[in] 返回值，如果指定时间的柱形图没有找到。如果*exact=false*，*iBarShift*返回最近柱形图的指数，其开仓时间小于指定时间（*time_open<time*）。如果这个柱形图没有找到（没有指定时间之前的历史记录），那么该函数返回 -1。如果*exact=true*，iBarShift不会搜索最近的柱形图而是立即返回 -1。

返回值

与指定时间对应的柱形图指数。如果与指定时间对应的柱形图没有找到（历史记录中有一个空白），那么函数返回 -1或最近柱形图的指数（取决于*'exact'*参数）。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 日期为星期日
   datetime time=D'2002.04.25 12:00';
   string symbol="GBPUSD";
   ENUM_TIMEFRAMES tf=PERIOD_H1;
   bool exact=false;
//--- 如果在指定时间没有柱形图，那么iBarShift将返回最近柱形图的指数。
   int bar_index=iBarShift(symbol,tf,time,exact);
//--- 调用iBarShift()之后，检查错误代码
   int error=GetLastError();
   if(error!=0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("iBarShift(): GetLastError=%d - The requested date %s "+
                  "for %s %s is not found in the available history",
                  error,TimeToString(time),symbol,EnumToString(tf));
      return;
     }
//--- iBarShift()函数被成功执行,返回结果exact=false
   PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),
               DayOfWeek(time),bar_index,string(exact));
   datetime bar_time=iTime(symbol,tf,bar_index);
   PrintFormat("Time of bar #%d is %s (%s)",
               bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
//--- 请求指定时间的柱形图指数;如果没有柱形图则将返回-1
   exact=true;
   bar_index=iBarShift(symbol,tf,time,exact);
//--- iBarShift()函数被成功执行,返回结果exact=true
   PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time)
               ,DayOfWeek(time),bar_index,string(exact));
  }
//+------------------------------------------------------------------+
//| 返回一周的每日名称                                               |
//+------------------------------------------------------------------+
string DayOfWeek(const datetime time)
  {
   MqlDateTime dt;
   string day="";
   TimeToStruct(time,dt);
   switch(dt.day_of_week)
     {
      case 0: day=EnumToString(SUNDAY);
      break;
      case 1: day=EnumToString(MONDAY);
      break;
      case 2: day=EnumToString(TUESDAY);
      break;
      case 3: day=EnumToString(WEDNESDAY);
      break;
      case 4: day=EnumToString(THURSDAY);
      break;
      case 5: day=EnumToString(FRIDAY);
      break;
      default:day=EnumToString(SATURDAY);
      break;
     }
//---
   return day;
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
/* Execution result
   1. GBPUSD PERIOD_H1 2018.06.10 12:00(SUNDAY): bar index is 64 (exact=false)
   Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
   2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY):bar index is -1 (exact=true)
*/
```

© 2000-2025, MetaQuotes Ltd.

# iClose

返回对应的图表上柱形图的收盘价（通过'shift'参数表示）。

```
double  iClose(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的收盘价（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyClose, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iHigh

返回对应的图表上柱形图的最高价（通过'shift'参数表示）。

```
double  iHigh(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的最高价（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyHigh, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iHighest

返回在对应图表上找到的最高值的指数（相对于当前柱形图的转移）。

```
int  iHighest(
   const string        symbol,              // 交易品种
   ENUM_TIMEFRAMES     timeframe,           // 周期
   ENUM_SERIESMODE     type,                // 时间序列标识符
   int                 count=WHOLE_ARRAY,   // 元素数量
   int                 start=0              // 指数
  );
```

参数

*交易品种*

[in] 将要执行搜索的交易品种。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*类型*

[in] 将要执行搜索的时间序列标识符。可以等于ENUM_SERIESMODE的任何值。

*count=WHOLE_ARRAY*

[in] 时间序列中的元素数量（从当前柱形图向指数递增的方向），搜索应在其中执行。

*start=0*

[in] 初始柱形图的指数（相对于当前柱形图的转移），从这里开始搜索最高的值。忽略负值并用0零代替。

返回值

在对应图表上找到的最高值的指数（相对于当前柱形图的转移）或者错误情况下为0。错误详情，请调用GetLastError()函数。

例如：

```
   double val;
//--- 在20个连续柱形图中计算最高收盘值
//--- 从指数4到指数23，在当前时间表
   int val_index=iHighest(NULL,0,MODE_CLOSE,20,4);
   if(val_index!=-1)
      val=High[val_index];
   else
      PrintFormat("iHighest() call error. Error code=%d",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

# iLow

返回对应的图表上柱形图的最低价（通过'shift'参数表示）。

```
double  iLow(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的最低价（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyLow, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iLowest

返回在对应图表上找到的最小值的指数（相对于当前柱形图的转移）。

```
int  iLowest(
   const string        symbol,              // 交易品种
   ENUM_TIMEFRAMES     timeframe,           // 周期
   ENUM_SERIESMODE     type,                // 时间序列标识符
   int                 count=WHOLE_ARRAY,   // 元素数量
   int                 start=0              // 指数
  );
```

参数

*交易品种*

[in] 将要执行搜索的交易品种。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*类型*

[in] 将要执行搜索的时间序列标识符。可以等于ENUM_SERIESMODE的任何值。

*count=WHOLE_ARRAY*

[in] 时间序列中的元素数量（从当前柱形图向指数递增的方向），搜索应在其中执行。

*start=0*

[in] 初始柱形图的指数（相对于当前柱形图的转移），从这里开始搜索最低的值。忽略负值并用0零代替。

返回值

在对应图表上找到的最低值的指数（相对于当前柱形图的转移）或者错误情况下为0。错误详情，请调用GetLastError()函数。

例如：

```
   double val;
//--- 在15个连续柱形图中搜索真实交易量的最低值的柱形图
//--- 从指数10到指数24，在当前时间表
   int val_index=iLowest(NULL,0,MODE_REAL_VOLUME,15,10);
   if(val_index!=-1)
      val=Low[val_index];
   else
      PrintFormat("iLowest() call error. Error code=%d",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

# iOpen

返回对应的图表上柱形图的开盘价（通过'shift'参数表示）。

```
double  iOpen(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的开盘价（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyOpen, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iTime

返回对应的图表上柱形图的开盘时间（通过'shift'参数表示）。

```
datetime  iTime(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的开盘时间（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 日期为星期日
   datetime time=D'2018.06.10 12:00';
   string symbol="GBPUSD";
   ENUM_TIMEFRAMES tf=PERIOD_H1;
   bool exact=false;
//--- 在指定时间没有柱形图，那么iBarShift将返回最近柱形图的指数
   int bar_index=iBarShift(symbol,tf,time,exact);
   PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,st
   datetime bar_time=iTime(symbol,tf,bar_index);
   PrintFormat("Time of bar #%d is %s (%s)",
               bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
```

© 2000-2025, MetaQuotes Ltd.

```
//PrintFormat(iTime(symbol,tf,bar_index));
//--- 请求指定时间的柱形图指数;但如果没有柱形图则将返回-1
   exact=true;
   bar_index=iBarShift(symbol,tf,time,exact);
   PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,st
  }
//+------------------------------------------------------------------+
//| 返回一周的每日名称                                               |
//+------------------------------------------------------------------+
string DayOfWeek(const datetime time)
  {
   MqlDateTime dt;
   string day="";
   TimeToStruct(time,dt);
   switch(dt.day_of_week)
     {
      case 0: day=EnumToString(SUNDAY);
      break;
      case 1: day=EnumToString(MONDAY);
      break;
      case 2: day=EnumToString(TUESDAY);
      break;
      case 3: day=EnumToString(WEDNESDAY);
      break;
      case 4: day=EnumToString(THURSDAY);
      break;
      case 5: day=EnumToString(FRIDAY);
      break;
      default:day=EnumToString(SATURDAY);
      break;
     }
//---
   return day;
  }
/* The result:
   1. GBPUSD PERIOD_H1 2018.06.10 12:00(SUNDAY): bar index is 64 (exact=false)
   Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
   2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY):bar index is -1 (exact=true)
*/
```

另见

CopyTime, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iTickVolume

返回对应的图表上柱形图的报价量（通过'shift'参数表示）。

```
long  iTickVolume(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的报价量（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyTickVolume, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iRealVolume

返回对应的图表上柱形图的真实交易量（通过'shift'参数表示）。

```
long  iRealVolume(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的真实交易量（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time   = iTime(Symbol(),Period(),shift);
   double   open   = iOpen(Symbol(),Period(),shift);
   double   high   = iHigh(Symbol(),Period(),shift);
   double   low    = iLow(Symbol(),Period(),shift);
   double   close  = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars   = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "   ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "   ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyRealVolume, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iVolume

返回对应的图表上柱形图的报价量（通过'shift'参数表示）。

```
long  iVolume(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的报价量（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time   = iTime(Symbol(),Period(),shift);
   double   open   = iOpen(Symbol(),Period(),shift);
   double   high   = iHigh(Symbol(),Period(),shift);
   double   low    = iLow(Symbol(),Period(),shift);
   double   close  = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars   = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "   ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "   ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopyTickVolume, CopyRates

© 2000-2025, MetaQuotes Ltd.

# iSpread

返回对应的图表上柱形图的点差值（通过'shift'参数表示）。

```
long  iSpread(
   const string         symbol,          // 交易品种
   ENUM_TIMEFRAMES      timeframe,       // 周期
   int                  shift            // 转移
   );
```

参数

*交易品种*

[in] 金融工具的交易品种名称。NULL表示当前交易品种。

*timeframe*

[in] 周期。它可以是ENUM_TIMEFRAMES其中一个枚举值。0表示当前图表周期。

*shift*

[in] 来自时间序列返回值的指数（相对于当前柱形图的指定柱形图数量的向后偏移）。

返回值

对应的图表上柱形图的点差值（通过'shift'参数表示）或者错误情况下为0。错误详情，请调用GetLastError()函数。

注意

这个函数始终返回实际数据。为此，它对每次调用期间指定交易品种/周期的时间序列执行请求。这意味着如果第一次函数调用时没有准备好的数据，则可能需要一些时间来准备结果。

这个函数不会存储之前的调用结果，也没有快速返回值的本地缓存。

例如：

```
input int shift=0;
//+------------------------------------------------------------------+
//| 函数-事件处理程序"tick"                                          |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
           "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
           "Open: "  ,DoubleToString(open,Digits()),"\n",
```

© 2000-2025, MetaQuotes Ltd.

```
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

另 见

CopySpread, CopyRates

© 2000-2025, MetaQuotes Ltd.

# 自定义交易品种

创建和编辑自定义交易品种属性的函数。

当程序端与某个交易服务器连接时，用户可以使用交易商所提供的金融交易品种的时间序列。可以使用的金融交易品种列表显示在市场报价窗口。不同的函数组可以接收交易品种属性的数据，交易时段和市场深度更新。

这部分描述的函数组可以创建自定义交易品种。为此，用户可以使用交易服务器现有的交易品种，文本文件或外部数据源。

| 函数 | 操作 |
| --- | --- |
| CustomSymbolCreate | 在指定组以指定名称创建一个自定义交易品种 |
| CustomSymbolDelete | 删除指定名称的自定义交易品种 |
| CustomSymbolSetInteger | 为自定义交易品种设置整型属性值 |
| CustomSymbolSetDouble | 为自定义交易品种设置真实型属性值 |
| CustomSymbolSetString | 为自定义交易品种设置字符串类型属性值 |
| CustomSymbolSetMarginRate | 按照订单类型和方向，为自定义交易品种设置预付款比率 |
| CustomSymbolSetSessionQuote | 为指定交易品种和工作日设置指定报价时段的起止时间 |
| CustomSymbolSetSessionTrade | 为指定交易品种和工作日设置指定交易时段的起止时间 |
| CustomRatesDelete | 删除指定时间间隔内自定义交易品种价格历史的全部柱形图 |
| CustomRatesReplace | 以MqlRates类型数组数据完全替换指定时间间隔内自定义交易品种的价格历史 |
| CustomRatesUpdate | 将丢失的柱形图添加到自定义交易品种历史并用MqlRates类型数组数据替换现有数据 |
| CustomTicksAdd | 添加MqlTick类型的数组数据到自定义交易品种的价格历史。自定义交易品种一定要在市场报价窗口来选择 |
| CustomTicksDelete | 删除指定时间间隔内自定义交易品种价格历史的全部报价 |
| CustomTicksReplace | 以MqlTick类型数组数据完全替换指定时间间隔内自定义交易品种的价格历史 |
| CustomBookAdd | 为自定义交易品种传递市场深度的状态 |

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolCreate

在指定组以指定名称创建一个自定义交易品种。

```
bool  CustomSymbolCreate(
   const string      symbol_name,          //自定义交易品种名称
   const string      symbol_path="",       // 将要创建交易品种的组的名称
   const string      symbol_origin=NULL    //交易品种的名称，该交易品种用作创建自定义交易品种的
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。它不应包含交易品种所在的组或分组。

*symbol_path=""*

[in] 交易品种所在的组名。

*symbol_origin=NULL*

[in] 交易品种的名称，已创建自定义交易品种的属性将在此复制。创建自定义交易品种之后，可以使用相应的函数将任何属性值更改为必要的属性值。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

所有自定义交易品种都在特殊的 "自定义"部分创建。如果没有指定组名 CustomSymbolCreate 函数中的*symbol_path* 参数包含空字符串或NULL)，那么自定义交易品种在 "自定义"根部分生成。在这里，我们可以与文件系统进行类比，其中组和子组可以被视为文件夹和子文件夹

交易品种和组的名称只能包含拉丁字母，不可带有标点符号，空格或特殊字符 (只可以包括以下符号".", "_", "&" 和 "#")。不建议使用下列字符<, >, :, ", /, \, |, ?, *。

无论创建所在的组名是什么，自定义交易品种名称都应该是独一无二的。如果已存在同名的交易品种，CustomSymbolCreate()函数返回 "false"，而之后的GetLastError()调用返回错误5300 (ERR_NOT_CUSTOM_SYMBOL)或5304 (ERR_CUSTOM_SYMBOL_EXIST)。

交易品种名称的总长度不应超过31个字符。否则，CustomSymbolCreate()返回"false"，并激活错误5302 -ERR_CUSTOM_SYMBOL_NAME_LONG。

*symbol_path*参数可以有两种设置方式：

- 只有组名而没有自定义交易品种的名称，例如 - "CFD\\Metals"。最好使用此选项来避免错误。
- 或者<组>名称 + 组分隔符"\\"+<自定义交易品种名称>，例如 -"CFD\\Metals\\Platinum"。在这种情况下，组名应该以自定义交易品种的确切名称结束。如果不匹配，自定义交易品种仍会被创建，但却不在预期的组中。例如，如果*symbol_path*="CFD\\Metals\\Platinum" 和*symbol_name*="platinum" (注册错误)，那么名为"platinum"的自定义交易品种会创建在"Custom\CFD\Metals\Platinum"组中。SymbolInfoGetString("platinum",SYMBOL_PATH)函数返回"Custom\CFD\Metals\Platinum\platinum"值。

© 2000-2025, MetaQuotes Ltd.

请注意，SYMBOL_PATH属性返回末位带有交易品种名称的路径。因此，如果您希望在完全相同的组中创建自定义交易品种，则无法在不更改的情况下复制它。在这种情况下，为了不得到上面描述的结果，需要删除交易品种名称。

如果不存在的交易品种被设置为*symbol_origin*参数，那么自定义交易品种创建为空，就好像未设置*symbol_origin*参数一样。这种情况下，激活错误4301 -ERR_MARKET_UNKNOWN_SYMBOL。

*symbol_path*参数长度不应超过127个字符，如果该参数在末尾指定，要考虑"Custom \\"，"\\"组分隔符和交易品种名称。

示例：

```
//+------------------------------------------------------------------+
//|                                              CustomSymbolCreate.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // 自定义交易品种所基于的交易品种名称

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 如果创建自定义交易品种失败，则在日志中通知
   if(!CustomSymbolCreate(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL_ORIGIN
     {
      Print("CustomSymbolCreate() failed. Error ", GetLastError());
      return;
     }

//--- 检查所创建交易品种是否已存在，并取得创建它所在的组
   bool   custom= false;
   bool   exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   string path  = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_PATH);

//--- 在日志中打印创建交易品种的结果和组的名称(指定并从交易品种属性中获得)
   PrintFormat("Custom symbol '%s' created\n"+
               "Symbol '%s' is exist: %s\n"+
               "Symbol '%s' is custom: %s\n"+
               "Path specified in the settings: '%s'\n"+
               "Path returned from the 'SYMBOL_PATH' property: '%s'",
               CUSTOM_SYMBOL_NAME,
```

© 2000-2025, MetaQuotes Ltd.

```
                CUSTOM_SYMBOL_NAME, (string)exist,
                CUSTOM_SYMBOL_NAME, (string)custom,
                CUSTOM_SYMBOL_PATH,
                path);

//--- 等待两秒再删除所创建的交易品种，在日志中打印结果消息
   Sleep(2000);
   ResetLastError();
   bool deleted = CustomSymbolDelete(CUSTOM_SYMBOL_NAME);
   Print(deleted ? StringFormat("Custom symbol '%s' removed", CUSTOM_SYMBOL_NAME) : St
   /*
   结果：
   Custom symbol 'EURUSD.C' created
   Symbol 'EURUSD.C' is exist: true
   Symbol 'EURUSD.C' is custom: true
   Path specified in the settings: 'Forex'
   Path returned from the 'SYMBOL_PATH' property: 'Custom\Forex\EURUSD.C'
   Custom symbol 'EURUSD.C' removed
   */
  }
```

另见

SymbolName, SymbolSelect, CustomSymbolDelete

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolDelete

删除指定名称的自定义交易品种。

```
bool  CustomSymbolDelete(
   const string      symbol_name          // 自定义交易品种名称
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。不应该与现有交易品种名称相同。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

不可以删除显示在市场报价或已打开图表的自定义交易品种。

示例：

```
//+------------------------------------------------------------------+
//|                                          CustomSymbolDelete.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // 自定义交易品种所基于的交易品种名称

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 如果创建自定义交易品种失败，则在日志中通知
   if(!CustomSymbolCreate(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL_ORIGIN
     {
      Print("CustomSymbolCreate() failed. Error ", GetLastError());
      return;
     }

//--- 检查要创建的交易品种是否存在并在日志中打印结果
   bool custom= false;
```

© 2000-2025, MetaQuotes Ltd.

```
   bool exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   PrintFormat("Custom symbol '%s' exists: %s", CUSTOM_SYMBOL_NAME, (string)exist);

//--- 等待两秒再删除所创建的交易品种，在日志中打印结果消息
   Sleep(2000);
   ResetLastError();
   bool deleted = CustomSymbolDelete(CUSTOM_SYMBOL_NAME);
   Print(deleted ? StringFormat("Custom symbol '%s' removed", CUSTOM_SYMBOL_NAME) : St

//--- 检查要创建的交易品种是否存在并在日志中打印结果
   exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   PrintFormat("Custom symbol '%s' exists: %s", CUSTOM_SYMBOL_NAME, (string)exist);
   /*
   结果：
   Custom symbol 'EURUSD.C' exists: true
   Custom symbol 'EURUSD.C' removed
   Custom symbol 'EURUSD.C' exists: false
   */
  }
```

另 见

SymbolName, SymbolSelect, CustomSymbolCreate

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetInteger

为自定义交易品种设置整型属性值。

```
bool  CustomSymbolSetInteger(
   const string               symbol_name,      // 交易品种名称
   ENUM_SYMBOL_INFO_INTEGER   property_id,      // 属性 ID
   long                       property_value    // 属性值
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*property_id*

[in] 交易品种属性ID。该值可以是ENUM_SYMBOL_INFO_INTEGER其中一个枚举值。

*property_value*

[in] 包含属性值的长整型变量。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

如果在交易品种规范中更改了任何这些属性，那么自定义交易品种的分钟和报价历史将被完全移除：

- SYMBOL_CHART_MODE -用于构建柱的价格类型（卖价或最后价）
- SYMBOL_DIGITS -价格显示中小数点后的小数位

删除自定义交易品种历史以后，程序端将使用更新的属性尝试创建一个新的历史。手动更改自定义交易品种属性时也会遇到同样的问题。

示例：

```
//+------------------------------------------------------------------+
//|                                      CustomSymbolSetInteger.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // 自定义交易品种所基于的交易品种名称

//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 脚本程序起始函数                                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 读取并在日志中打印自定义交易品种所基于交易品种的属性
//--- （交易模式，止损单价格水平和交易冻结水平）
   ENUM_SYMBOL_TRADE_EXECUTION origin_exe_mode = (ENUM_SYMBOL_TRADE_EXECUTION)SymbolIn
   int origin_stops_level = (int)SymbolInfoInteger(CUSTOM_SYMBOL_ORIGIN, SYMBOL_TRADE_
   int origin_freeze_level= (int)SymbolInfoInteger(CUSTOM_SYMBOL_ORIGIN, SYMBOL_TRADE_

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Deal execution mode: %s\n  Stops Level: %d\n  Freeze Level: %d",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               StringSubstr(EnumToString(origin_exe_mode), 23), origin_stops_level, or

//--- 设置自定义交易品种属性的其它数值
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_EXEMODE, SYMBOL_TRADE
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_STOPS_LEVEL, 10);
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_FREEZE_LEVEL, 3);

//--- 如果在设置属性时出错，就在日志中显示相应消息
   if(!res)
      Print("CustomSymbolSetInteger() failed. Error ", GetLastError());

//--- 读取并在日志中打印修改过的自定义交易品种属性
//--- （交易模式，止损单价格水平和交易冻结水平）
   ENUM_SYMBOL_TRADE_EXECUTION custom_exe_mode = (ENUM_SYMBOL_TRADE_EXECUTION)SymbolIn
   int custom_stops_level = (int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_ST
   int custom_freeze_level= (int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_FR

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Deal execution mode: %s\n  Stops Level: %d\n  Freeze Level: %d",
               CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_ORIGIN,
               StringSubstr(EnumToString(custom_exe_mode), 23), custom_stops_level, cu

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex

//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Sleep(16);
       //--- 当按下 Del 时, 删除所创建的自定义交易品种
       if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
         {
          if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
             PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
          break;
         }
      }
//--- 退出前清空图表
   Comment("");
   /*
   结果:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Deal execution mode: INSTANT
     Stops Level: 0
     Freeze Level: 0
   Custom symbol 'EURUSD.C' based on 'EURUSD':
     Deal execution mode: MARKET
     Stops Level: 10
     Freeze Level: 3
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
```

另见

SymbolInfoInteger

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetDouble

为自定义交易品种设置真实型属性值。

```
bool  CustomSymbolSetDouble(
   const string              symbol_name,      // 交易品种名称
   ENUM_SYMBOL_INFO_DOUBLE   property_id,      // 属性ID
   double                    property_value    // 属性值
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*property_id*

[in] 交易品种属性ID。该值可以是ENUM_SYMBOL_INFO_DOUBLE其中一个枚举值。

*property_value*

[in] 包含属性值的双精度类型变量。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

如果在交易品种规范中更改了任何这些属性，那么自定义交易品种的分钟和报价历史将被完全移除：

- SYMBOL_POINT -一个点的值
- SYMBOL_TRADE_TICK_SIZE -指定可允许的最小价格变化的报价值
- SYMBOL_TRADE_TICK_VALUE -一个盈利持仓的一个报价变动值

删除自定义交易品种历史以后，程序端将使用更新的属性尝试创建一个新的历史。手动更改自定义交易品种属性时也会遇到同样的问题。

示例：

```
//+------------------------------------------------------------------+
//|                                        CustomSymbolSetDouble.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd.   |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // 自定义交易品种所基于的交易品种名称
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0 (创建交易品种成功) 也不为 5304 (交易品种已经被创建) - 退出
   if(create!=0 && create!=5304)
      return;

//--- 读取并在日志中打印自定义交易品种所基于交易品种的属性
//--- (最小交易量,最大交易量,执行交易的最小交易量改变步长)
   double origin_vol_min = SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_MIN);
   double origin_vol_max = SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_MAX);
   double origin_vol_step= SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_STEP);

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Volume Min: %.2f\n  Volume Max: %.2f\n  Volume Step: %.2f",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               origin_vol_min, origin_vol_max, origin_vol_step);

//--- 设置自定义交易品种属性的其它数值
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MIN, 0.1);
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MAX, 1000);
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_STEP, 0.1);

//--- 如果在设置属性时出错,就在日志中显示相应消息
   if(!res)
      Print("CustomSymbolSetDouble() failed. Error ", GetLastError());

//--- 读取并在日志中打印修改过的自定义交易品种属性
//--- (最小交易量,最大交易量,执行交易的最小交易量改变步长)
   double custom_vol_min = SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MIN);
   double custom_vol_max = SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MAX);
   double custom_vol_step= SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_STEP);

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Volume Min: %.2f\n  Volume Max: %.2f\n  Volume Step: %.2f",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               custom_vol_min, custom_vol_max, custom_vol_step);

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex

//--- 在无尽循环中等待按下 Esc 或 Del 键
```

© 2000-2025, MetaQuotes Ltd.

```
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Volume Min: 0.01
     Volume Max: 500.00
     Volume Step: 0.01
   Custom symbol 'EURUSD' based on 'EURUSD.C':
     Volume Min: 0.10
     Volume Max: 1000.00
     Volume Step: 0.10
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种，返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304，在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- 成功
   return(true);
  }
```

另 见

SymbolInfoDouble

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetString

为自定义交易品种设置字符串类型属性值。

```
bool  CustomSymbolSetString(
   const string               symbol_name,      // 交易品种名称
   ENUM_SYMBOL_INFO_STRING    property_id,      // 属性 ID
   string                     property_value    // 属性值
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*property_id*

[in] 交易品种属性ID。该值可以是ENUM_SYMBOL_INFO_STRING其中一个枚举值。

*property_value*

[in] 包含属性值的字符串类型变量。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

如果在交易品种规范中更改了SYMBOL_FORMULA属性（设定自定义交易品种价格结构的公式），那么自定义交易品种的分钟和报价历史将被完全移除。删除自定义交易品种历史以后，程序端将使用新公式尝试创建一个新的历史。手动更改自定义交易品种公式时也会遇到同样的问题。

示例：

```
//+------------------------------------------------------------------+
//|                                        CustomSymbolSetString.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd.   |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // 自定义交易品种所基于的交易品种名称

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 读取并在日志中打印自定义交易品种所基于交易品种的属性
//--- （最小交易量，最大交易量，执行交易的最小交易量改变步长）
   string origin_basis    = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_BASIS);
   string origin_category = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_CATEGORY);
   string origin_formula  = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_FORMULA);

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Basis: %s\n  Category: %s\n  Formula: %s",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               origin_basis, origin_category, origin_formula);

//--- 设置自定义交易品种属性的其它数值
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_BASIS, CUSTOM_SYMBOL_ORIGIN);
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_CATEGORY, "FX");
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_FORMULA, ("1.0 / "+CUSTOM_SY

//--- 如果在设置属性时出错，就在日志中显示相应消息
   if(!res)
      Print("CustomSymbolSetString() failed. Error ", GetLastError());

//--- 读取并在日志中打印修改过的自定义交易品种属性
//--- （最小交易量，最大交易量，执行交易的最小交易量改变步长）
   string custom_basis    = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_BASIS);
   string custom_category = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_CATEGORY);
   string custom_formula  = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_FORMULA);

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Basis: %s\n  Category: %s\n  Formula: %s",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               custom_basis, custom_category, custom_formula);

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex

//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种
```

© 2000-2025, MetaQuotes Ltd.

```
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Basis:
     Category:
     Formula:
   Custom symbol 'EURUSD' based on 'EURUSD.C':
     Basis: EURUSD
     Category: FX
     Formula: 1.0 / EURUSD
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种，返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304，在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
       return(false);
      }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
      {
       PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
       return(false);
      }
//--- 成功
   return(true);
  }
```

另见

SymbolInfoString

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetMarginRate

按照订单类型和方向，为自定义交易品种设置预付款比率。

```
bool  CustomSymbolSetMarginRate(
   const string       symbol_name,              // 交易品种名称
   ENUM_ORDER_TYPE    order_type,               // 订单类型
   double             initial_margin_rate,      // 初始预付款比率
   double             maintenance_margin_rate   // 维持预付款比率
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*order_type*

[in] 订单类型。

*initial_margin_rate*

[in] 具有初始预付款比率的双精度类型变量。初始预付款是适当方向1手交易的保证金。乘以初始预付款比率后，我们会获得指定类型订单下单账户上保留的资金数额。

*maintenance_margin_rate*

[in] 具有维持预付款比率的双精度类型变量。维持预付款是适当方向保持1手持仓的最低金额。乘以维持预付款比率后，我们会获得激活指定类型订单后账户上保留的资金数额。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

示例：

```
//+------------------------------------------------------------------+
//|                                    CustomSymbolSetMarginRate.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd.   |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME      Symbol()+".C"  // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH      "Forex"        // 创建交易品种的所在组名称
#define   CUSTOM_SYMBOL_ORIGIN    Symbol()       // 自定义交易品种所基于的交易品种名称

#define   INITIAL_MARGIN_RATE       1.5          // 初始保证金比率
#define   MAINTENANCE_MARGIN_RATE   1.5          // 维持保证金比率

//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 脚本程序起始函数                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 读取并在日志中打印自定义交易品种所基于交易品种的属性
//--- （买入和卖出订单的初始和维持保证金比率）
   double initial_margin_rate_buy=0;
   double maintenance_margin_rate_buy=0;
   double initial_margin_rate_sell=0;
   double maintenance_margin_rate_sell=0;

   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_ORIGIN, ORDER_TYPE_BUY, initial_margin_rate_b
      return;
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_ORIGIN, ORDER_TYPE_SELL,initial_margin_rate_s
      return;

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Buy order initial margin rate: %f\n  Buy order maintenance margin ra
               "  Sell order initial margin rate: %f\n  Sell order maintenance margin
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               initial_margin_rate_buy, maintenance_margin_rate_buy, initial_margin_ra

//--- 设置自定义交易品种属性的其它数值
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_BUY, INITIAL_MARGIN_
   res &=CustomSymbolSetMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_SELL,INITIAL_MARGIN_

//--- 如果在设置属性时出错，就在日志中显示相应消息
   if(!res)
      Print("CustomSymbolSetMarginRate() failed. Error ", GetLastError());

//--- 读取并在日志中打印修改过的自定义交易品种属性
//--- （买入和卖出订单的初始和维持保证金比率）
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_BUY, initial_margin_rate_buy
      return;
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_SELL,initial_margin_rate_sel
      return;

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Buy order initial margin rate: %f\n  Buy order maintenance margin ra
               "  Sell order initial margin rate: %f\n  Sell order maintenance margin
```

© 2000-2025, MetaQuotes Ltd.

```
                CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_ORIGIN,
                initial_margin_rate_buy, maintenance_margin_rate_buy, initial_margin_ra

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex

//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时, 删除所创建的自定义交易品种
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Buy order initial margin rate: 1.000000
     Buy order maintenance margin rate: 0.000000
     Sell order initial margin rate: 1.000000
     Sell order maintenance margin rate: 0.000000
   Custom symbol 'EURUSD.C' based on 'EURUSD':
     Buy order initial margin rate: 1.500000
     Buy order maintenance margin rate: 1.500000
     Sell order initial margin rate: 1.500000
     Sell order maintenance margin rate: 1.500000
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
```

© 2000-2025, MetaQuotes Ltd.

```
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
      }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 返回保证金比率                                                   |
//+------------------------------------------------------------------+
bool GetSymbolMarginRate(const string symbol, const ENUM_ORDER_TYPE order_type, double
  {
   ResetLastError();
   if(!SymbolInfoMarginRate(symbol, order_type, initial_margin_rate, maintenance_margi
     {
      PrintFormat("%s: SymbolInfoMarginRate(%s, %s) failed. Error %d",__FUNCTION__, sy
      return false;
     }
   return true;
  }
```

另见

SymbolInfoMarginRate

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetSessionQuote

为指定交易品种和工作日设置指定报价时段的起止时间。

```
bool  CustomSymbolSetSessionQuote(
   const string      symbol_name,          // 交易品种名称
   ENUM_DAY_OF_WEEK  day_of_week,          // 工作日
   uint              session_index,        // 时段索引
   datetime          from,                 // 开始时段
   datetime          to                    // 结束时段
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*ENUM_DAY_OF_WEEK*

[in] 工作日，来自ENUM_DAY_OF_WEEK枚举值。

*uint*

[in]时段索引，设置起止时间。时段索引从0开始。

*from*

[in] 时段按照00:00的格式以秒为单位计算开始时间，忽略变量中的数据值。

*to*

[in] 时段按照00:00的格式以秒为单位计算结束时间，忽略变量中的数据值。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

如果*session_index*指定的时段已经存在，那么该函数只需编辑时段的起止时间。

如果这个时段(*from=0* and *to=0*)通过了起止参数均为零，那么将会删除*session_index*适当的时段，而时段索引向下移动。

只能按照顺序添加时段。换句话说，只有0索引时段存在时您才能添加*session_index=1*。如果打破这个规则，则不会创建新时段，而函数本身也会返回'false'。

示例：

```
//+------------------------------------------------------------------+
//|                                  CustomSymbolSetSessionQuote.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"            // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"                  // 要创建交易品种的组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()                 // 自定义交易品种所基于交易品种的

#define   SESSION_0_FROM         D'1970.01.01 00:15:00'   // 时段 0 起始时间
#define   SESSION_0_TO           D'1970.01.01 11:59:00'   // 时段 0 结束时间
#define   SESSION_1_FROM         D'1970.01.01 12:15:00'   // 时段 1 起始时间
#define   SESSION_1_TO           D'1970.01.01 23:59:00'   // 时段 1 结束时间

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 使用基础交易品种和时段索引打印标头并且
//--- 按每周一到周五每天循环打印报价时段的起始和结束时间
   for(int session=0; session<2; session++)
     {
      PrintFormat("Quote session %d of '%s' symbol from which the custom '%s' was crea
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_ORIGIN, (ENUM_DAY_OF_WEEK)day_of_we
     }

//--- 在两个时段中循环
   bool res=true;
   for(int session=0; session<2; session++)
     {
      datetime from = SESSION_0_FROM;
      datetime to   = SESSION_0_TO;
      if(session>0)
        {
         from = SESSION_1_FROM;
         to   = SESSION_1_TO;
        }
      //--- 按自定义交易品种在一周中的每天设置报价时段
      ResetLastError();
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         res &=CustomSymbolSetSessionQuote(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果设置任何时段出错，就在日志中显示对应消息
   if(!res)
      Print("CustomSymbolSetSessionQuote() failed. Error ", GetLastError());

//--- 使用自定义交易品种和时段索引打印标头并且
//--- 按每周一到周五每天循环打印报价时段的起始和结束时间
   for(int session=0; session<2; session++)
     {
      PrintFormat("Quote session %d of custom symbol '%s' based on '%s'", session, CUS
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_of_week
     }

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   Quote session 0 of 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     00:15 - 23:55
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   Quote session 1 of 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     Session not set
   - Tuesday    Session not set
   - Wednesday  Session not set
   - Thursday   Session not set
   - Friday     Session not set
   Quote session 0 of custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     00:15 - 11:59
   - Tuesday    00:15 - 11:59
   - Wednesday  00:15 - 11:59
   - Thursday   00:15 - 11:59
   - Friday     00:15 - 11:59
```

© 2000-2025, MetaQuotes Ltd.

```
   Quote session 1 of custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     12:15 - 23:59
   - Tuesday    12:15 - 23:59
   - Wednesday  12:15 - 23:59
   - Thursday   12:15 - 23:59
   - Friday     12:15 - 23:59
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败,在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
```

© 2000-2025, MetaQuotes Ltd.

```
    return(true);
   }
//+------------------------------------------------------------------+
//| 发送指定报价时段的起始和结束时间                                 |
//| 用于指定的交易品种和一周中的某天到日志中                         |
//+------------------------------------------------------------------+
void SymbolInfoSessionQuotePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
   {
//--- 声明用于记录报价时段起始和结束时间的变量
    datetime date_from;  // session start time
    datetime date_to;    // session end time

//--- 根据枚举常数创建一周中一天的名称
    string week_day=EnumToString(day_of_week);
    if(week_day.Lower())
       week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- 根据交易品种和一周中的某天取得报价时段数据
    if(!SymbolInfoSessionQuote(symbol, day_of_week, session_index, date_from, date_to))
      {
       int err=GetLastError();
       string message=(err==4307 ? StringFormat("- %-10s Session not set", week_day) :
                       StringFormat("SymbolInfoSessionQuote(%s, %s, session %d) failed
       Print(message);
       return;
      }

//--- 把指定的报价时段数据发送到日志中
    PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Tir
   }
```

另见

SymbolInfoSessionQuote, Symbol info, TimeToStruct, Date structure

© 2000-2025, MetaQuotes Ltd.

# CustomSymbolSetSessionTrade

为指定交易品种和工作日设置指定交易时段的起止时间

```
bool  CustomSymbolSetSessionTrade(
   const string      symbol_name,          // 交易品种名称
   ENUM_DAY_OF_WEEK  day_of_week,          // 工作日
   uint              session_index,        // 时段索引
   datetime          from,                 // 开始时段
   datetime          to                    // 结束时段
   );
```

参数

*symbol_name*

[in] 自定义交易品种名称。

*ENUM_DAY_OF_WEEK*

[in] 工作日，来自ENUM_DAY_OF_WEEK枚举值。

*uint*

[in]时段索引，设置起止时间。时段索引从0开始。

*from*

[in] 时段按照00:00的格式以秒为单位计算开始时间，忽略变量中的数据值。

*to*

[in] 时段按照00:00的格式以秒为单位计算结束时间，忽略变量中的数据值。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

如果*session_index*指定的时段已经存在，那么该函数只需编辑时段的起止时间。

如果这个时段(*from=0* and *to=0*)通过了起止参数均为零，那么将会删除*session_index*适当的时段，而时段索引向下移动。

只能按照顺序添加时段。换句话说，只有0索引时段存在时您才能添加*session_index=1*。如果打破这个规则，则不会创建新时段，而函数本身也会返回'false'。

示例：

```
//+------------------------------------------------------------------+
//|                                  CustomSymbolSetSessionTrade.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"           // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"                 // 要创建交易品种的组名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()                // 自定义交易品种所基于交易品种的

#define   SESSION_0_FROM         D'1970.01.01 00:15:00'  // 时段 0 起始时间
#define   SESSION_0_TO           D'1970.01.01 11:59:00'  // 时段 0 结束时间
#define   SESSION_1_FROM         D'1970.01.01 12:15:00'  // 时段 1 起始时间
#define   SESSION_1_TO           D'1970.01.01 23:59:00'  // 时段 1 结束时间

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 使用基础交易品种和时段索引打印标头并且
//--- 按照一周中从星期一至星期五循环在日志中打印每个交易时段的起始和结束时间
   for(int session=0; session<2; session++)
     {
      PrintFormat("Trade session %d of the '%s' symbol from which the custom '%s' was
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_ORIGIN, (ENUM_DAY_OF_WEEK)day_of_we
     }

//--- 在两个时段中循环
   bool res=true;
   for(int session=0; session<2; session++)
     {
      datetime from = SESSION_0_FROM;
      datetime to   = SESSION_0_TO;
      if(session>0)
        {
         from = SESSION_1_FROM;
         to   = SESSION_1_TO;
        }
      //--- 按自定义交易品种在一周中的每天设置报价时段
      ResetLastError();
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         res &=CustomSymbolSetSessionQuote(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果设置任何时段出错，就在日志中显示对应消息
   if(!res)
      Print("CustomSymbolSetSessionTrade() failed. Error ", GetLastError());

//--- 使用自定义交易品种和时段索引打印标头并且
//--- 按照一周中从星期一至星期五循环在日志中打印每个交易时段的起始和结束时间
   for(int session=0; session<2; session++)
     {
      PrintFormat("Trade session %d of a custom symbol '%s' based on '%s'", session, C
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_of_week
     }

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   Trade session 0 of the 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     00:15 - 23:55
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   Trade session 1 of the 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     Session not set
   - Tuesday    Session not set
   - Wednesday  Session not set
   - Thursday   Session not set
   - Friday     Session not set
   Trade session 0 of a custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     00:15 - 11:59
   - Tuesday    00:15 - 11:59
   - Wednesday  00:15 - 11:59
   - Thursday   00:15 - 11:59
   - Friday     00:15 - 11:59
```

© 2000-2025, MetaQuotes Ltd.

```
   Trade session 1 of a custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     12:15 - 23:59
   - Tuesday    12:15 - 23:59
   - Wednesday  12:15 - 23:59
   - Thursday   12:15 - 23:59
   - Friday     12:15 - 23:59
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败,在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
```

© 2000-2025, MetaQuotes Ltd.

```
    return(true);
  }
//+------------------------------------------------------------------+
//| 发送指定报价时段的起始和结束时间                                 |
//| 用于指定的交易品种和一周中的某天到日志中                         |
//+------------------------------------------------------------------+
void SymbolInfoSessionTradePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- 声明用于记录报价时段起始和结束时间的变量
   datetime date_from;  // session start time
   datetime date_to;    // session end time

//--- 根据枚举常数创建一周中一天的名称
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- 根据交易品种和一周中的某天取得报价时段数据
   if(!SymbolInfoSessionTrade(symbol, day_of_week, session_index, date_from, date_to))
     {
      int err=GetLastError();
      string message=(err==4307 ? StringFormat("- %-10s Session not set", week_day) :
                      StringFormat("SymbolInfoSessionTrade(%s, %s, session %d) failed
      Print(message);
      return;
     }

//--- 把指定交易时段的数据发送到日志中
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Tir
  }
```

另见

SymbolInfoSessionTrade, Symbol info, TimeToStruct, Date structure

© 2000-2025, MetaQuotes Ltd.

# CustomRatesDelete

删除指定时间间隔内自定义交易品种价格历史的全部柱形图。

```
int  CustomRatesDelete(
   const string      symbol,        // 交易品种名称
   datetime          from,          // 开始日期
   datetime          to             // 结束日期
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*from*

[in] 即将移除的指定范围内价格历史的第一个柱形图时间。

*to*

[in] 即将移除的指定范围内价格历史的最后柱形图时间。

返回值

已删除柱形图的数量，错误情况下为-1。

示例：

```
//+------------------------------------------------------------------+
//|                                              CustomRatesDelete.mq5 |
//|                                 Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"      // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"            // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()           // 自定义交易品种的基础交易品种名称

#define   DATARATES_COUNT        4                  // 发送到日志的柱形数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果错误代码不为0 (创建交易品种成功) 也不为 5304 (交易品种已经被创建) - 退出
   if(create!=0 && create!=5304)
      return;

//--- 获取标准交易品种的柱数
   int bars=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);

//--- 把标准交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_OR
      return;
     }

//--- 把复制的数据设为自定义交易品种的分钟历史
   ResetLastError();
   if(CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates)<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- 在更新历史数据之后,读取自定义交易品种的柱形数量
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- 把自定义交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- 把自定义交易品种1分钟历史中的最新 DATARATES_COUNT 个柱的数据打印到日志中
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- 删除自定义交易品种1分钟历史中从倒数第二个柱开始的两个柱的数据
   datetime time_from= rates[bars-3].time;
   datetime time_to  = rates[bars-2].time;
   ResetLastError();
   int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, time_from, time_to);
   if(deleted<0)
     {
      PrintFormat("CustomRatesDelete(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 在删除两个柱的历史数据之后，再次读取自定义交易品种的柱数
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- 再次读取自定义交易品种1分钟时段所有剩余柱的数据
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- 在日志中打印更新后的自定义交易品种1分钟历史中的最后 DATARATES_COUNT 个柱
   PrintFormat("\nLast %d bars after applying CustomRatesDelete() with %d deleted bars
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   Last 4 bars of the custom symbol's minute history:
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 20:53:00 1.07341 1.07347 1.07336 1.07343            38        0
```

© 2000-2025, MetaQuotes Ltd.

```
   [1] 2024.06.18 20:54:00 1.07344 1.07354 1.07344 1.07353            21         0
   [2] 2024.06.18 20:55:00 1.07353 1.07362 1.07351 1.07356            32         0
   [3] 2024.06.18 20:56:00 1.07356 1.07358 1.07352 1.07354            24         0

   Last 4 bars after applying CustomRatesDelete() with 2 deleted bars:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 20:51:00 1.07357 1.07358 1.07347 1.07349            25         0
   [1] 2024.06.18 20:52:00 1.07349 1.07350 1.07336 1.07341            31         0
   [2] 2024.06.18 20:53:00 1.07341 1.07347 1.07336 1.07343            38         0
   [3] 2024.06.18 20:56:00 1.07356 1.07358 1.07352 1.07354            24         0
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败,在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
```

另见

CustomRatesReplace, CustomRatesUpdate, CopyRates

© 2000-2025, MetaQuotes Ltd.

# CustomRatesReplace

以MqlRates类型数组数据完全替换指定时间间隔内自定义交易品种的价格历史。

```
int  CustomRatesReplace(
   const string      symbol,              // 交易品种名称
   datetime          from,                // 开始日期
   datetime          to,                  // 结束日期
   const MqlRates&   rates[],             // 即将用于自定义交易品种的数据数组
   uint              count=WHOLE_ARRAY    // 即将使用的rates[]数组元素的数量
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*从*

[in] 即将更新的指定范围内价格历史的第一个柱形图时间。

*到*

[in] 即将更新的指定范围内价格历史的最后柱形图时间。

*rates[]*

[in] M1的MqlRates类型历史数据数组。

*count=WHOLE_ARRAY*

[in] 即将用于替换的*rates[]*数组元素的数量。WHOLE_ARRAY意味着所有*rates[]*数组元素都应被用于替换。

返回值

已更新柱形图的数量，错误情况下为-1。

注意

如果*rates[]*数组的柱形图超出指定范围，则忽略该柱形图。如果价格历史已经存在这个柱形图并输入指定范围，则它可被替换。指定范围内以外当前价格历史的其他所有柱形图保持不变。有关高开低收价格应该是正确的*rates[]*数组数据，而开盘时间柱形图应该对应M1 时间周期。

示例：

```
//+------------------------------------------------------------------+
//|                                          CustomRatesReplace.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

© 2000-2025, MetaQuotes Ltd.

```
#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"      // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"            // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()           // 自定义交易品种的基础交易品种名称

#define   DATARATES_COUNT        4                  // 发送到日志的柱形数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0 (创建交易品种成功) 也不为 5304 (交易品种已经被创建) - 退出
   if(create!=0 && create!=5304)
      return;

//--- 获取标准交易品种的柱数
   int bars=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);

//--- 把标准交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_OR
      return;
     }

//--- 把复制的数据设为自定义交易品种的分钟历史
   ResetLastError();
   if(CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates)<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- 在更新历史数据之后,读取自定义交易品种的柱形数量
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- 把自定义交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 把自定义交易品种1分钟历史中的最新 DATARATES_COUNT 个柱的数据打印到日志中
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- 修改自定义交易品种1分钟历史中从倒数第二个柱开始的两个柱的数据
   datetime time_from= rates[bars-3].time;
   datetime time_to  = rates[bars-2].time;

//--- 使从倒数第二个柱开始的两个柱的所有价格等于这两个柱在 'rates' 数组中的开盘价格
   rates[bars-3].high=rates[bars-3].open;
   rates[bars-3].low=rates[bars-3].open;
   rates[bars-3].close=rates[bars-3].open;

   rates[bars-2].high=rates[bars-2].open;
   rates[bars-2].low=rates[bars-2].open;
   rates[bars-2].close=rates[bars-2].open;

//--- 使用修改过的 'rates' 数组中的数据替换已有柱的数据
   ResetLastError();
   int replaced=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(replaced<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- 在修改了两个柱的历史数据后，再次读取自定义交易品种的柱数
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- 再次读取自定义交易品种分钟时段柱的全部数据
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- 在日志中打印更新后的自定义交易品种1分钟历史中的最后 DATARATES_COUNT 个柱
   PrintFormat("\nLast %d bars after applying CustomRatesUpdate() with %d replaced bar
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 当按下 Del 时, 删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
      }
//--- 退出前清空图表
   Comment("");
   /*
   结果:
   Last 4 bars of the custom symbol's minute history:
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.07.29 13:37:00 1.08394 1.08396 1.08388 1.08390            16        1
   [1] 2024.07.29 13:38:00 1.08389 1.08400 1.08389 1.08398            35        1
   [2] 2024.07.29 13:39:00 1.08398 1.08410 1.08394 1.08410            29        1
   [3] 2024.07.29 13:40:00 1.08409 1.08414 1.08408 1.08414            14        1

   Last 4 bars after applying CustomRatesUpdate() with 250820 replaced bars:
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.07.29 13:37:00 1.08394 1.08396 1.08388 1.08390            16        1
   [1] 2024.07.29 13:38:00 1.08389 1.08389 1.08389 1.08389            35        1
   [2] 2024.07.29 13:39:00 1.08398 1.08398 1.08398 1.08398            29        1
   [3] 2024.07.29 13:40:00 1.08409 1.08414 1.08408 1.08414            14        1
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
```

另见

CustomRatesDelete, CustomRatesUpdate, CopyRates

© 2000-2025, MetaQuotes Ltd.

# CustomRatesUpdate

将丢失的柱形图添加到自定义交易品种历史并用MqlRates类型数组数据替换现有数据。

```
int  CustomRatesUpdate(
   const string     symbol,              //自定义交易品种名称
   const MqlRates&  rates[],             // 即将用于自定义交易品种的数据数组
   uint             count=WHOLE_ARRAY    // 即将使用的rates[]数组元素的数量
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*rates[]*

[in] M1的MqlRates类型历史数据数组。

*count=WHOLE_ARRAY*

[in] 即将用于更新的*rates[]*数组元素的数量。WHOLE_ARRAY意味着所有*rates[]*数组元素都应被使用。

返回值

已更新柱形图的数量，错误情况下为-1。

注意

如果当前自定义交易品种的历史中没有*rates[]*数组柱形图，则需要添加该柱形图。如果已存在这个柱形图，则它可被替换。当前价格历史的其他所有柱形图保持不变。有关高开低收价格应该是正确的*rates[]*数组数据，而开盘时间柱形图应该对应M1时间周期。

示例：

```
//+------------------------------------------------------------------+
//|                                            CustomRatesUpdate.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd.   |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"      // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"            // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()           // 自定义交易品种的基础交易品种名称

#define   DATARATES_COUNT        4                  // 发送到日志的柱形数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0 (创建交易品种成功) 也不为 5304 (交易品种已经被创建) - 退出
   if(create!=0 && create!=5304)
      return;

//--- 读取并在日志中打印标准交易品种柱形数量
   int bars_origin=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);
   PrintFormat("The symbol '%s' from which the custom '%s' was created has %d bars of

//--- 读取并在日志中打印自定义交易品种柱形数量
   int bars_custom=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);
   PrintFormat("Custom symbol '%s' created from symbol '%s' has %d bars of minute hist

//--- 把标准交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars_origin, rates)!=bars_origin)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_OR
      return;
     }

//--- 把复制的数据设为自定义交易品种的分钟历史
   ResetLastError();
   int updated=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(updated<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- 在添加历史后读取并在日志中打印自定义交易品种的柱形数量
   bars_custom=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);
   PrintFormat("\nAfter CustomRatesUpdate(), the custom symbol '%s' has %d bars of mi

//--- 把自定义交易品种1分钟时段的所有柱形数据读取至 MqlRates 数组
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars_custom, rates)!=bars_custom)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- 在日志中打印自定义交易品种分钟历史的最后四个柱形
```

© 2000-2025, MetaQuotes Ltd.

```
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars_custom-DATARATES_COUNT, DATARATES_COUNT);

//--- 把 MqlRates 数组中的数据替换成计算后的数据，公式为 1.0 / SymbolName
   for(int i=0; i<bars_custom; i++)
     {
      rates[i].open  =(rates[i].open !=0  ? 1.0 / rates[i].open  : rates[i].open);
      rates[i].high  =(rates[i].high !=0  ? 1.0 / rates[i].high  : rates[i].high);
      rates[i].low   =(rates[i].low  !=0  ? 1.0 / rates[i].low   : rates[i].low);
      rates[i].close =(rates[i].close!=0  ? 1.0 / rates[i].close : rates[i].close);
     }

//--- 把修改过的数据设置为自定义交易品种的分钟历史
   ResetLastError();
   updated=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(updated<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- 再次把自定义交易品种分钟时段的所有柱形数据读取至 MqlRates 数组
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars_custom, rates)!=bars_custom)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- 在日志中打印更新过的自定义交易品种分钟历史中的最后四个柱形
   Print("\nLast %d bars after changing the custom symbol calculation formula:", DATAR
   ArrayPrint(rates, digits, NULL, bars_custom-DATARATES_COUNT, DATARATES_COUNT);

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
```

© 2000-2025, MetaQuotes Ltd.

```
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果:
   The symbol 'EURUSD' from which the custom 'EURUSD.C' was created has 250488 bars of
   Custom symbol 'EURUSD.C' created from symbol 'EURUSD' has 0 bars of minute history

   After CustomRatesUpdate(), the custom symbol 'EURUSD.C' has 250488 bars of minute h
   Last 4 bars of the custom symbol's minute history:
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 11:14:00 1.07235 1.07239 1.07232 1.07239            24        0
   [1] 2024.06.18 11:15:00 1.07238 1.07239 1.07232 1.07235            44        0
   [2] 2024.06.18 11:16:00 1.07234 1.07238 1.07227 1.07234            37        0
   [3] 2024.06.18 11:17:00 1.07234 1.07234 1.07217 1.07225            41        0

   Last 4 bars after changing the custom symbol calculation formula:
                     [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 11:14:00 0.93253 0.93250 0.93256 0.93250            24        0
   [1] 2024.06.18 11:15:00 0.93251 0.93250 0.93256 0.93253            44        0
   [2] 2024.06.18 11:16:00 0.93254 0.93251 0.93260 0.93254            37        0
   [3] 2024.06.18 11:17:00 0.93254 0.93254 0.93269 0.93262            41        0
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
```

另见

CustomRatesReplace, CustomRatesDelete, CopyRates

© 2000-2025, MetaQuotes Ltd.

# CustomTicksAdd

添加MqlTick类型的数组数据到自定义交易品种的价格历史。自定义交易品种一定要在市场报价窗口来选择。

```
int  CustomTicksAdd(
   const string     symbol,              // 交易品种名称
   const MqlTick&   ticks[],             // 应该应用于自定义交易品种的报价数据数组
   uint             count=WHOLE_ARRAY    // 即将使用的ticks[]数组元素的数量
   );
```

参数

*交易品种*

[in] 自定义交易品种的名称。

*ticks[]*

[in] MqlTick类型的报价数据数组按照从早期到最近的顺序排列，例如ticks[k].time_msc <= ticks[n].time_msc，if k<n。

*count=WHOLE_ARRAY*

[in] 即将用于添加的*ticks[]*数组元素的数量。WHOLE_ARRAY意味着所有*ticks[]*数组元素都应被使用。

返回值

添加报价的数量，错误情况下为-1。

进一步注释

CustomTicksAdd函数只能用于在市场报价窗口打开的自定义交易品种。如果没有在市场报价窗口选择交易品种，您应该使用CustomTicksReplace添加报价。

CustomTicksAdd函数允许按照在交易商服务器上交付的模式传递报价。数据发送至市场报价来代替直接写入报价数据库。然后程序端将来自市场报价的价格保存到数据库。如果函数调用期间传递的数据量过于庞大，函数处理会发生变化以减少资源使用率。如果传递超过256个报价，数据将被分为两个部分。第一部分，即数据量大的部分直接写入报价数据库（如同CustomTicksReplace函数的操作）。第二部分包含128个报价，被传递到市场报价窗口，程序端在这里将报价保存到数据库。

MqlTick结构有两个时间值字段：time（以秒表示的报价时间）和time_msc（以毫秒表示的报价时间），从1970年1月1日开始计算。添加报价的这些字段按照以下顺序进行处理：

1. 如果ticks[k].time_msc!=0，我们用它来填写ticks[k].time字段，例如为报价设置ticks[k].time=ticks[k].time_msc/1000（整除）
2. 如果ticks[k].time_msc==0并且ticks[k].time!=0，那么将乘以1000来获得毫秒表示的时间，例如ticks[k].time_msc=ticks[k].time*1000
3. 如果ticks[k].time_msc==0和ticks[k].time==0，那么将调用CustomTicksAdd函数时当前交易服务器的毫秒时间写入这些字段。

如果ticks[k].bid，ticks[k].ask，ticks[k].last或ticks[k].volume的数值大于零，那么将适当的标识组合写入ticks[k].flags字段：

- TICK_FLAG_BID -报价已更改卖价
- TICK_FLAG_ASK -报价已更改买价

© 2000-2025, MetaQuotes Ltd.

- TICK_FLAG_LAST -报价已更改最后交易价
- TICK_FLAG_VOLUME -报价已更改交易量

如果字段的值小于或等于零，对应的标识将不写入ticks[k].flags字段。

自定义交易品种历史记录不添加TICK_FLAG_BUY and TICK_FLAG_SELL标识。

示例：

```
//+------------------------------------------------------------------+
//|                                              CustomTicksAdd.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"           // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // 自定义交易品种的基础交易品种名称

#define   DATATICKS_TO_COPY      UINT_MAX          // 复制的报价数量
#define   DATATICKS_TO_PRINT     20                // 发送到日志中的报价数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 ||
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0 （创建交易品种成功） 也不为 5304 （交易品种已经被创建） - 退出
   if(create!=0 && create!=5304)
      return;

//--- 把标准交易品种的报价数据读取至 MqlTick 数组中
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- 打印接收到的标准交易品种的第一个和最后一个报价的时间
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把标准交易品种最后一个报价的 DATATICKS_TO_PRINT 数据打印至日志中
```

© 2000-2025, MetaQuotes Ltd.

```
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRINT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在市场报价窗口中添加一个自定义交易品种
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- 把报价数组数据添加到自定义交易品种的价格历史中
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", ar
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", add

//--- 把自定义交易品种报价数据读取至 MqlTick 数组中
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- 打印收到的自定义交易品种第一个和最后一个报价的时间
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把自定义交易品种最后报价的 DATATICKS_TO_PRINT 数据打印到日志中
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 当按下 Del 时, 删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果:
   Requested 4294967295 ticks to download tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351183943 tic
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.20 21:18:12.010

   The last 20 ticks for the standard symbol 'EURUSD':
     351183923th Tick: 2024.06.20 21:17:46.380 Bid=1.07124 (Info tick)
     351183924th Tick: 2024.06.20 21:17:47.779 Ask=1.07125 Bid=1.07125 (Info tick)
     351183925th Tick: 2024.06.20 21:17:48.584 Ask=1.07124 Bid=1.07124 (Info tick)
     351183926th Tick: 2024.06.20 21:17:49.481 Ask=1.07125 (Info tick)
     351183927th Tick: 2024.06.20 21:17:49.985 Ask=1.07122 Bid=1.07122 (Info tick)
     351183928th Tick: 2024.06.20 21:17:50.482 Ask=1.07124 Bid=1.07124 (Info tick)
     351183929th Tick: 2024.06.20 21:17:51.584 Ask=1.07123 Bid=1.07123 (Info tick)
     351183930th Tick: 2024.06.20 21:17:52.786 Ask=1.07124 Bid=1.07124 (Info tick)
     351183931th Tick: 2024.06.20 21:17:53.487 Ask=1.07125 Bid=1.07125 (Info tick)
     351183932th Tick: 2024.06.20 21:17:53.989 Ask=1.07126 Bid=1.07126 (Info tick)
     351183933th Tick: 2024.06.20 21:17:55.789 Ask=1.07125 Bid=1.07125 (Info tick)
     351183934th Tick: 2024.06.20 21:17:58.495 Ask=1.07126 Bid=1.07126 (Info tick)
     351183935th Tick: 2024.06.20 21:18:00.102 Bid=1.07126 (Info tick)
     351183936th Tick: 2024.06.20 21:18:00.698 Ask=1.07129 Bid=1.07129 (Info tick)
     351183937th Tick: 2024.06.20 21:18:03.699 Bid=1.07129 (Info tick)
     351183938th Tick: 2024.06.20 21:18:04.699 Ask=1.07128 Bid=1.07128 (Info tick)
     351183939th Tick: 2024.06.20 21:18:05.901 Ask=1.07129 Bid=1.07129 (Info tick)
     351183940th Tick: 2024.06.20 21:18:07.606 Ask=1.07128 Bid=1.07128 (Info tick)
     351183941th Tick: 2024.06.20 21:18:11.512 Ask=1.07127 Bid=1.07127 (Info tick)
     351183942th Tick: 2024.06.20 21:18:12.010 Ask=1.07126 Bid=1.07126 (Info tick)
   ...
```

© 2000-2025, MetaQuotes Ltd.

```
   Start of adding 351183943 ticks to the history of the custom symbol 'EURUSD.C'
   Added 351183943 ticks to the history of the custom symbol 'EURUSD.C' in 269890 ms
   ...
   Requested 351183943 ticks to download tick history for the symbol 'EURUSD.C'
   The tick history for the 'EURUSD.C' symbol is received in the amount of 351183943 t
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.20 21:18:12.010

   The last 20 ticks for the custom symbol 'EURUSD.C':
     351183923th Tick: 2024.06.20 21:17:46.380 Ask=1.07124 Bid=1.07124 (Info tick)
     351183924th Tick: 2024.06.20 21:17:47.779 Ask=1.07125 Bid=1.07125 (Info tick)
     351183925th Tick: 2024.06.20 21:17:48.584 Ask=1.07124 Bid=1.07124 (Info tick)
     351183926th Tick: 2024.06.20 21:17:49.481 Ask=1.07125 Bid=1.07125 (Info tick)
     351183927th Tick: 2024.06.20 21:17:49.985 Ask=1.07122 Bid=1.07122 (Info tick)
     351183928th Tick: 2024.06.20 21:17:50.482 Ask=1.07124 Bid=1.07124 (Info tick)
     351183929th Tick: 2024.06.20 21:17:51.584 Ask=1.07123 Bid=1.07123 (Info tick)
     351183930th Tick: 2024.06.20 21:17:52.786 Ask=1.07124 Bid=1.07124 (Info tick)
     351183931th Tick: 2024.06.20 21:17:53.487 Ask=1.07125 Bid=1.07125 (Info tick)
     351183932th Tick: 2024.06.20 21:17:53.989 Ask=1.07126 Bid=1.07126 (Info tick)
     351183933th Tick: 2024.06.20 21:17:55.789 Ask=1.07125 Bid=1.07125 (Info tick)
     351183934th Tick: 2024.06.20 21:17:58.495 Ask=1.07126 Bid=1.07126 (Info tick)
     351183935th Tick: 2024.06.20 21:18:00.102 Ask=1.07126 Bid=1.07126 (Info tick)
     351183936th Tick: 2024.06.20 21:18:00.698 Ask=1.07129 Bid=1.07129 (Info tick)
     351183937th Tick: 2024.06.20 21:18:03.699 Ask=1.07129 Bid=1.07129 (Info tick)
     351183938th Tick: 2024.06.20 21:18:04.699 Ask=1.07128 Bid=1.07128 (Info tick)
     351183939th Tick: 2024.06.20 21:18:05.901 Ask=1.07129 Bid=1.07129 (Info tick)
     351183940th Tick: 2024.06.20 21:18:07.606 Ask=1.07128 Bid=1.07128 (Info tick)
     351183941th Tick: 2024.06.20 21:18:11.512 Ask=1.07127 Bid=1.07127 (Info tick)
     351183942th Tick: 2024.06.20 21:18:12.010 Ask=1.07126 Bid=1.07126 (Info tick)
   */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种，返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304，在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败,在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 把指定数量的报价读取到数组中                                     |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- 通知开始载入历史数据
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, sy

//--- 为接收报价做3次尝试
   int attempts=0;
   while(attempts<3)
     {
      //--- 测量接收报价的开始时间
      uint start=GetTickCount();

      //--- 从 1970.01.01 00:00.001 (参数 from=1 ms) 开始请求报价历史
      int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
      if(received!=-1)
        {
         //--- 显示报价数量和花费的时间
         PrintFormat("The tick history for the '%s' symbol is received in the amount

         //--- 如果报价历史已同步, 错误代码等于0 - 返回 'true'
         if(GetLastError()==0)
            return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
         PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms
                     symbol, received, GetTickCount()-start, GetLastError());
        }
      //--- 计数尝试次数
      attempts++;
      //--- 暂停一秒以等待报价数据库同步结束
      Sleep(1000);
     }
//--- 3次尝试复制报价失败
   return(false);
  }
//+------------------------------------------------------------------+
//| 返回报价的字符串类型描述                                         |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTES

//--- 检查报价标志
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)    == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)   == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)    == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)    == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)   == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- 首先检查用于交易的报价标志（没有 CustomTicksAdd()）
   if(buy_tick || sell_tick)
     {
      //--- 构建交易报价的输出内容
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tick
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tick
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
      desc += "(Trade tick)";
     }
   else
     {
      //--- 构建信息报价的输出稍有不同
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- 返回报价描述
   return(desc);
  }
```

© 2000-2025, MetaQuotes Ltd.

另见

CustomRatesDelete, CustomRatesUpdate, CustomTicksReplace, CopyTicks, CopyTicksRange

© 2000-2025, MetaQuotes Ltd.

# CustomTicksDelete

删除指定时间间隔内自定义交易品种价格历史的全部报价。

```
int  CustomTicksDelete(
   conststring       symbol,           // 交易品种名称
   long              from_msc,         // 开始日期
   long              to_msc            // 结束日期
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*from_msc*

[in] 即将移除的指定范围内价格历史的第一个报价时间。时间以毫秒为单位从01.01.1970开始计算。

*to_msc*

[in] 即将移除的指定范围内价格历史的最后报价时间。时间以毫秒为单位从01.01.1970开始计算。

返回值

已删除报价的数量，错误情况下为-1。

示例：

```
//+------------------------------------------------------------------+
//|                                              CustomTicksDelete.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                           https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"      // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"            // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()           // 自定义交易品种的基础交易品种名称

#define   DATATICKS_TO_COPY      UINT_MAX           // 复制的报价数量
#define   DATATICKS_TO_DELETE    10                 // 删除报价的数量
#define   DATATICKS_TO_PRINT     20                 // 发送到日志中的报价数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 把标准交易品种的报价数据读取至 MqlTick 数组中
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- 打印接收到的标准交易品种的第一个和最后一个报价的时间
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把标准交易品种最后一个报价的 DATATICKS_TO_PRINT 数据打印至日志中
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRINT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在市场报价窗口中添加一个自定义交易品种
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- 把报价数组数据添加到自定义交易品种的价格历史中
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", ar
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", add

//--- 把新增的自定义交易品种报价数据读取至 MqlTick 数组中
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- 打印收到的自定义交易品种第一个和最后一个报价的时间
```

© 2000-2025, MetaQuotes Ltd.

```
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把自定义交易品种最后报价的 DATATICKS_TO_PRINT 数据打印到日志中
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 读取将从历史中删除的报价时间，以毫秒为单位
   long time_from=array[total-DATATICKS_TO_DELETE-1].time_msc;

//--- 在自定义交易品种数组中的最新的报价中删除 DATATICKS_TO_DELETE 个报价
   Print("...");
   start=GetTickCount();
   PrintFormat("Start deleting %u ticks in the history of the custom symbol '%s'", DAT
   int deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, time_from, array[total-2].time_ms
   PrintFormat("Deleted %u ticks in the history of the custom symbol '%s' in %u ms", d

//--- 把最新修改过的自定义交易品种报价数据读取至 MqlTick 数组中
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- 打印删除自定义交易品种报价范围中的第一个和最后一个报价的时间
   total=(int)array.Size();
   PrintFormat("Time of the first tick from the changed history: %s.%03u, Time of the
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把自定义交易品种最后报价的 DATATICKS_TO_PRINT 数据打印到日志中
   PrintFormat("\nThe last %d ticks of custom symbol '%s' with modified history:", DAT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      Sleep(16);
      //--- 当按下 Del 时，删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   Requested 4294967295 ticks to get tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351199027 tic
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 10:10:40.392

   The last 20 ticks for the standard symbol 'EURUSD':
     351199007th Tick: 2024.06.21 10:10:23.045 Bid=1.07032 (Info tick)
     351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
     351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 (Info tick)
     351199010th Tick: 2024.06.21 10:10:25.146 Bid=1.07032 (Info tick)
     351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
     351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
     351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
     351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
     351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
     351199016th Tick: 2024.06.21 10:10:30.258 Bid=1.07036 (Info tick)
     351199017th Tick: 2024.06.21 10:10:30.872 Ask=1.07035 Bid=1.07035 (Info tick)
     351199018th Tick: 2024.06.21 10:10:31.358 Ask=1.07036 (Info tick)
     351199019th Tick: 2024.06.21 10:10:31.859 Ask=1.07037 Bid=1.07037 (Info tick)
     351199020th Tick: 2024.06.21 10:10:32.377 Ask=1.07039 Bid=1.07039 (Info tick)
     351199021th Tick: 2024.06.21 10:10:32.962 Ask=1.0704 Bid=1.0704 (Info tick)
     351199022th Tick: 2024.06.21 10:10:33.961 Ask=1.07039 Bid=1.07039 (Info tick)
     351199023th Tick: 2024.06.21 10:10:34.667 Ask=1.0704 (Info tick)
     351199024th Tick: 2024.06.21 10:10:35.170 Bid=1.0704 (Info tick)
     351199025th Tick: 2024.06.21 10:10:38.266 Ask=1.07041 Bid=1.07041 (Info tick)
     351199026th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
```

© 2000-2025, MetaQuotes Ltd.

```
...
Start of adding 351199027 ticks to the history of the custom symbol 'EURUSD.C'
Added 351199027 ticks to the history of the custom symbol 'EURUSD.C' in 261594 ms
...
Requested 351199027 ticks to get tick history for the symbol 'EURUSD.C'
The tick history for the 'EURUSD.C' symbol is received in the amount of 351199027 t
First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 10:10:40.392

The last 20 ticks for the custom symbol 'EURUSD.C':
  351199007th Tick: 2024.06.21 10:10:23.045 Ask=1.07032 Bid=1.07032 (Info tick)
  351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
  351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 Bid=1.07032 (Info tick)
  351199010th Tick: 2024.06.21 10:10:25.146 Ask=1.07032 Bid=1.07032 (Info tick)
  351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
  351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
  351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
  351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
  351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
  351199016th Tick: 2024.06.21 10:10:30.258 Ask=1.07036 Bid=1.07036 (Info tick)
  351199017th Tick: 2024.06.21 10:10:30.872 Ask=1.07035 Bid=1.07035 (Info tick)
  351199018th Tick: 2024.06.21 10:10:31.358 Ask=1.07036 Bid=1.07036 (Info tick)
  351199019th Tick: 2024.06.21 10:10:31.859 Ask=1.07037 Bid=1.07037 (Info tick)
  351199020th Tick: 2024.06.21 10:10:32.377 Ask=1.07039 Bid=1.07039 (Info tick)
  351199021th Tick: 2024.06.21 10:10:32.962 Ask=1.0704 Bid=1.0704 (Info tick)
  351199022th Tick: 2024.06.21 10:10:33.961 Ask=1.07039 Bid=1.07039 (Info tick)
  351199023th Tick: 2024.06.21 10:10:34.667 Ask=1.0704 Bid=1.0704 (Info tick)
  351199024th Tick: 2024.06.21 10:10:35.170 Ask=1.0704 Bid=1.0704 (Info tick)
  351199025th Tick: 2024.06.21 10:10:38.266 Ask=1.07041 Bid=1.07041 (Info tick)
  351199026th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
...
Start deleting 10 ticks in the history of the custom symbol 'EURUSD.C'
Deleted 10 ticks in the history of the custom symbol 'EURUSD.C' in 188 ms
...
Requested 351199027 ticks to get tick history for the symbol 'EURUSD.C'
The tick history for the 'EURUSD.C' symbol is received in the amount of 351199017 t
Time of the first tick from the changed history: 2011.12.19 00:00:08.000, Time of t

The last 20 ticks of custom symbol 'EURUSD.C' with modified history:
  351198997th Tick: 2024.06.21 10:10:14.935 Ask=1.07036 Bid=1.07036 (Info tick)
  351198998th Tick: 2024.06.21 10:10:15.533 Ask=1.07035 Bid=1.07035 (Info tick)
  351198999th Tick: 2024.06.21 10:10:17.736 Ask=1.07036 Bid=1.07036 (Info tick)
  351199000th Tick: 2024.06.21 10:10:18.540 Ask=1.07037 Bid=1.07037 (Info tick)
  351199001th Tick: 2024.06.21 10:10:19.046 Ask=1.07038 Bid=1.07038 (Info tick)
  351199002th Tick: 2024.06.21 10:10:19.542 Ask=1.07036 Bid=1.07036 (Info tick)
  351199003th Tick: 2024.06.21 10:10:20.041 Ask=1.07035 Bid=1.07035 (Info tick)
  351199004th Tick: 2024.06.21 10:10:21.041 Ask=1.07035 Bid=1.07035 (Info tick)
  351199005th Tick: 2024.06.21 10:10:21.544 Ask=1.07032 Bid=1.07032 (Info tick)
  351199006th Tick: 2024.06.21 10:10:22.344 Ask=1.07032 Bid=1.07032 (Info tick)
  351199007th Tick: 2024.06.21 10:10:23.045 Ask=1.07032 Bid=1.07032 (Info tick)
```

© 2000-2025, MetaQuotes Ltd.

```
      351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
      351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 Bid=1.07032 (Info tick)
      351199010th Tick: 2024.06.21 10:10:25.146 Ask=1.07032 Bid=1.07032 (Info tick)
      351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
      351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
      351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
      351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
      351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
      351199016th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
    */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304, 在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                            |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败,在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 把指定数量的报价读取到数组中                                     |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- 通知开始载入历史数据
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, sy

//--- 为接收报价做3次尝试
   int attempts=0;
   while(attempts<3)
     {
      //--- 测量接收报价的开始时间
      uint start=GetTickCount();

      //--- 从 1970.01.01 00:00.001 (参数 from=1 ms) 开始请求报价历史
      int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
      if(received!=-1)
        {
         //--- 显示报价数量和花费的时间
         PrintFormat("The tick history for the '%s' symbol is received in the amount o

         //--- 如果报价历史已同步, 错误代码等于0 - 返回 'true'
         if(GetLastError()==0)
            return(true);

         PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms.
                     symbol, received, GetTickCount()-start, GetLastError());
        }
      //--- 计数尝试次数
      attempts++;
      //--- 暂停一秒以等待报价数据库同步结束
      Sleep(1000);
     }
//--- 3次尝试复制报价失败
   return(false);
  }
//+------------------------------------------------------------------+
//| 返回报价的字符串类型描述                                         |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTES
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 检查报价标志
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)    == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)   == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)    == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)    == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)   == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- 首先检查用于交易的报价标志（没有 CustomTicksAdd()）
   if(buy_tick || sell_tick)
     {
      //--- 构建交易报价的输出内容
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tick
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tick
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
      desc += "(Trade tick)";
     }
   else
     {
      //--- 构建信息报价的输出稍有不同
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- 返回报价描述
   return(desc);
  }
```

另见

CustomRatesDelete, CustomRatesUpdate, CustomTicksReplace, CopyTicks, CopyTicksRange

© 2000-2025, MetaQuotes Ltd.

# CustomTicksReplace

以MqlTick类型数组数据完全替换指定时间间隔内自定义交易品种的价格历史。

```
int  CustomTicksReplace(
   const string      symbol,           // 交易品种名称
   long              from_msc,         // 开始日期
   long              to_msc,           // 结束日期
   const MqlTick&    ticks[],          // 即将用于自定义交易品种的数据数组
   uint              count=WHOLE_ARRAY // 即将使用的ticks[]数组元素的数量
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*from_msc*

[in] 即将移除的指定范围内价格历史的第一个报价时间。时间以毫秒为单位从01.01.1970开始计算。

*to_msc*

[in] 即将移除的指定范围内价格历史的最后报价时间。时间以毫秒为单位从01.01.1970开始计算。

*ticks[]*

[in] 以升序时间顺序排列的MqlTick类型报价数据数组。

*count=WHOLE_ARRAY*

[in] 即将用于在指定时间间隔内进行替换的*ticks[]*数组元素的数量。WHOLE_ARRAY意味着所有*ticks[]*数组元素都应被使用。

返回值

已更新报价的数量，错误情况下为-1。

注意

由于多个报价常常在报价流中有最多1毫秒的相同时间（正确报价时间保存在MqlTick结构的*time_msc*文件里），CustomTicksReplace函数不会根据时间自动排序*ticks[]*数组元素。因此，报价数组一定要及时以升序提前排列。

报价会日复一日的持续被替换，直至到达*to_msc*指定的时间或出现错误为止。从指定范围的第一天开始处理，然后是第二天。当检测到报价时间和升序（非降序）不匹配时，那么当日的报价替换停止。如果之前全部报价被成功替换，那么当日（错误报价时间）和指定间隔内全部剩余日保持不变。

如果*ticks[]*数组包含任何时间的非报价数据（通常是任何时间间隔），那么使用*ticks[]*报价数据后，丢失数据对应的"hole"显示在自定义交易品种的历史中。换句话说，调用丢失报价的CustomTicksReplace函数等同于删除部分报价历史，就好比调用"hole"间隔的CustomTicksDelete函数。

如果报价数据库在指定时间间隔内没有数据，那么CustomTicksReplace将添加到数据库报价构成的ticks[]数组。

© 2000-2025, MetaQuotes Ltd.

CustomTicksReplace函数通过报价数据库直接工作。

示例：

```
//+------------------------------------------------------------------+
//|                                          CustomTicksReplace.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // 自定义交易品种名称
#define   CUSTOM_SYMBOL_PATH     "Forex"           // 创建的交易品种所在组的名称
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // 自定义交易品种的基础交易品种名称

#define   DATATICKS_TO_COPY      UINT_MAX          // 复制的报价数量
#define   DATATICKS_TO_PRINT     20                // 发送到日志中的报价数量

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得创建自定义交易品种时的错误代码
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL

//--- 如果错误代码不为0（创建交易品种成功）也不为 5304（交易品种已经被创建）- 退出
   if(create!=0 && create!=5304)
      return;

//--- 把标准交易品种的报价数据读取至 MqlTick 数组中
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- 打印接收到的标准交易品种的第一个和最后一个报价的时间
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把标准交易品种最后一个报价的 DATATICKS_TO_PRINT 数据打印至日志中
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRINT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在市场报价窗口中添加一个自定义交易品种
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- 把报价数组数据添加到自定义交易品种的价格历史中
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", ar
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", add

//--- 把新增的自定义交易品种报价数据读取至 MqlTick 数组中
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- 打印收到的自定义交易品种第一个和最后一个报价的时间
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 把自定义交易品种最后报价的 DATATICKS_TO_PRINT 数据打印到日志中
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- 现在使用公式 Ask(Symbol) = 1.0 / Ask(Symbol), Bid(Symbol) = 1.0 / Bid(Symbol) 来在
   for(int i=0; i<total; i++)
     {
      array[i].ask = (array[i].ask !=0 ? 1.0 / array[i].ask : array[i].ask);
      array[i].bid = (array[i].bid !=0 ? 1.0 / array[i].bid : array[i].bid);
     }
   Print("\nNow the ticks are changed");

//--- 使用修改过的报价数组数据替换自定义交易品种的报价历史
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("...");
   start=GetTickCount();
   PrintFormat("Start replacing %u changed ticks in the history of the custom symbol
   int replaced=CustomTicksReplace(CUSTOM_SYMBOL_NAME, array[0].time_msc, array[total-
   PrintFormat("Replaced %u ticks in the history of the custom symbol '%s' in %u ms",

//--- 把替换过的自定义交易品种报价数据读取至 MqlTick 数组中
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- 打印修改过的自定义交易品种报价的第一个和最后一个报价的时间
   total=(int)array.Size();
   PrintFormat("First changed tick time: %s.%03u, Last changed tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- 在日志中打印自定义交易品种最近 DATATICKS_TO_PRINT 个修改过的报价
   PrintFormat("\nThe last %d changed ticks for the custom symbol '%s':", DATATICKS_TO
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Changed tick: %s", i, GetTickDescription(array[i]));
     }

//--- 在图表注释区显示脚本终止键的提示
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- 在无尽循环中等待按下 Esc 或 Del 键
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- 当按下 Del 时, 删除所创建的自定义交易品种和它的数据
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- 删除柱形数据
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully d

         //--- 删除报价数据
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)


         //--- 删除交易品种
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
```

© 2000-2025, MetaQuotes Ltd.

```
         }
      }
//--- 退出前清空图表
   Comment("");
   /*
   结果：
   Requested 4294967295 ticks to get tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351195822 tic
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 08:39:03.113

   The last 20 ticks for the standard symbol 'EURUSD':
     351195802th Tick: 2024.06.21 08:38:10.076 Ask=1.07194 (Info tick)
     351195803th Tick: 2024.06.21 08:38:13.162 Ask=1.07195 (Info tick)
     351195804th Tick: 2024.06.21 08:38:13.872 Bid=1.07195 (Info tick)
     351195805th Tick: 2024.06.21 08:38:14.866 Ask=1.07194 Bid=1.07194 (Info tick)
     351195806th Tick: 2024.06.21 08:38:17.374 Bid=1.07194 (Info tick)
     351195807th Tick: 2024.06.21 08:38:18.883 Bid=1.07194 (Info tick)
     351195808th Tick: 2024.06.21 08:38:19.771 Bid=1.07194 (Info tick)
     351195809th Tick: 2024.06.21 08:38:20.873 Ask=1.07195 Bid=1.07195 (Info tick)
     351195810th Tick: 2024.06.21 08:38:22.278 Ask=1.07196 Bid=1.07196 (Info tick)
     351195811th Tick: 2024.06.21 08:38:22.775 Bid=1.07196 (Info tick)
     351195812th Tick: 2024.06.21 08:38:23.477 Bid=1.07196 (Info tick)
     351195813th Tick: 2024.06.21 08:38:38.194 Ask=1.07197 (Info tick)
     351195814th Tick: 2024.06.21 08:38:38.789 Ask=1.07196 (Info tick)
     351195815th Tick: 2024.06.21 08:38:39.290 Ask=1.07197 (Info tick)
     351195816th Tick: 2024.06.21 08:38:43.695 Ask=1.07196 (Info tick)
     351195817th Tick: 2024.06.21 08:38:52.203 Ask=1.07195 Bid=1.07195 (Info tick)
     351195818th Tick: 2024.06.21 08:38:55.105 Ask=1.07196 Bid=1.07196 (Info tick)
     351195819th Tick: 2024.06.21 08:38:57.607 Ask=1.07195 Bid=1.07195 (Info tick)
     351195820th Tick: 2024.06.21 08:39:00.512 Ask=1.07196 Bid=1.07196 (Info tick)
     351195821th Tick: 2024.06.21 08:39:03.113 Ask=1.07195 Bid=1.07195 (Info tick)
   ...
   Start of adding 351195822 ticks to the history of the custom symbol 'EURUSD.C'
   Added 351195822 ticks to the history of the custom symbol 'EURUSD.C' in 349407 ms
   ...
   Requested 351195822 ticks to get tick history for the symbol 'EURUSD.C'
   The tick history for the 'EURUSD.C' symbol is received in the amount of 351195822 t
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 08:39:03.113

   The last 20 ticks for the custom symbol 'EURUSD.C':
     351195802th Tick: 2024.06.21 08:38:10.076 Ask=1.07194 Bid=1.07194 (Info tick)
     351195803th Tick: 2024.06.21 08:38:13.162 Ask=1.07195 Bid=1.07195 (Info tick)
     351195804th Tick: 2024.06.21 08:38:13.872 Ask=1.07195 Bid=1.07195 (Info tick)
     351195805th Tick: 2024.06.21 08:38:14.866 Ask=1.07194 Bid=1.07194 (Info tick)
     351195806th Tick: 2024.06.21 08:38:17.374 Ask=1.07194 Bid=1.07194 (Info tick)
     351195807th Tick: 2024.06.21 08:38:18.883 Ask=1.07194 Bid=1.07194 (Info tick)
     351195808th Tick: 2024.06.21 08:38:19.771 Ask=1.07194 Bid=1.07194 (Info tick)
     351195809th Tick: 2024.06.21 08:38:20.873 Ask=1.07195 Bid=1.07195 (Info tick)
     351195810th Tick: 2024.06.21 08:38:22.278 Ask=1.07196 Bid=1.07196 (Info tick)
```

© 2000-2025, MetaQuotes Ltd.

```
      351195811th Tick: 2024.06.21 08:38:22.775 Ask=1.07196 Bid=1.07196 (Info tick)
      351195812th Tick: 2024.06.21 08:38:23.477 Ask=1.07196 Bid=1.07196 (Info tick)
      351195813th Tick: 2024.06.21 08:38:38.194 Ask=1.07197 Bid=1.07197 (Info tick)
      351195814th Tick: 2024.06.21 08:38:38.789 Ask=1.07196 Bid=1.07196 (Info tick)
      351195815th Tick: 2024.06.21 08:38:39.290 Ask=1.07197 Bid=1.07197 (Info tick)
      351195816th Tick: 2024.06.21 08:38:43.695 Ask=1.07196 Bid=1.07196 (Info tick)
      351195817th Tick: 2024.06.21 08:38:52.203 Ask=1.07195 Bid=1.07195 (Info tick)
      351195818th Tick: 2024.06.21 08:38:55.105 Ask=1.07196 Bid=1.07196 (Info tick)
      351195819th Tick: 2024.06.21 08:38:57.607 Ask=1.07195 Bid=1.07195 (Info tick)
      351195820th Tick: 2024.06.21 08:39:00.512 Ask=1.07196 Bid=1.07196 (Info tick)
      351195821th Tick: 2024.06.21 08:39:03.113 Ask=1.07195 Bid=1.07195 (Info tick)

    Now the ticks are changed
    ...
    Start replacing 351195822 changed ticks in the history of the custom symbol 'EURUSD
    Replaced 351195822 ticks in the history of the custom symbol 'EURUSD.C' in 452266 m
    ...
    Requested 351195822 ticks to get tick history for the symbol 'EURUSD.C'
    The tick history for the 'EURUSD.C' symbol is received in the amount of 351195822 t
    First changed tick time: 2011.12.19 00:00:08.000, Last changed tick time: 2024.06.2

    The last 20 changed ticks for the custom symbol 'EURUSD.C':
      351195802th Changed tick: 2024.06.21 08:38:10.076 Ask=0.93289 Bid=0.93289 (Info t
      351195803th Changed tick: 2024.06.21 08:38:13.162 Ask=0.93288 Bid=0.93288 (Info t
      351195804th Changed tick: 2024.06.21 08:38:13.872 Ask=0.93288 Bid=0.93288 (Info t
      351195805th Changed tick: 2024.06.21 08:38:14.866 Ask=0.93289 Bid=0.93289 (Info t
      351195806th Changed tick: 2024.06.21 08:38:17.374 Ask=0.93289 Bid=0.93289 (Info t
      351195807th Changed tick: 2024.06.21 08:38:18.883 Ask=0.93289 Bid=0.93289 (Info t
      351195808th Changed tick: 2024.06.21 08:38:19.771 Ask=0.93289 Bid=0.93289 (Info t
      351195809th Changed tick: 2024.06.21 08:38:20.873 Ask=0.93288 Bid=0.93288 (Info t
      351195810th Changed tick: 2024.06.21 08:38:22.278 Ask=0.93287 Bid=0.93287 (Info t
      351195811th Changed tick: 2024.06.21 08:38:22.775 Ask=0.93287 Bid=0.93287 (Info t
      351195812th Changed tick: 2024.06.21 08:38:23.477 Ask=0.93287 Bid=0.93287 (Info t
      351195813th Changed tick: 2024.06.21 08:38:38.194 Ask=0.93286 Bid=0.93286 (Info t
      351195814th Changed tick: 2024.06.21 08:38:38.789 Ask=0.93287 Bid=0.93287 (Info t
      351195815th Changed tick: 2024.06.21 08:38:39.290 Ask=0.93286 Bid=0.93286 (Info t
      351195816th Changed tick: 2024.06.21 08:38:43.695 Ask=0.93287 Bid=0.93287 (Info t
      351195817th Changed tick: 2024.06.21 08:38:52.203 Ask=0.93288 Bid=0.93288 (Info t
      351195818th Changed tick: 2024.06.21 08:38:55.105 Ask=0.93287 Bid=0.93287 (Info t
      351195819th Changed tick: 2024.06.21 08:38:57.607 Ask=0.93288 Bid=0.93288 (Info t
      351195820th Changed tick: 2024.06.21 08:39:00.512 Ask=0.93287 Bid=0.93287 (Info t
      351195821th Changed tick: 2024.06.21 08:39:03.113 Ask=0.93288 Bid=0.93288 (Info t
    */
  }
//+------------------------------------------------------------------+
//| 创建一个自定义交易品种, 返回错误代码                                |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义自定义交易品种所基于的交易品种名称
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- 如果创建自定义交易品种失败并且错误代码不是 5304，在日志中报告
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- 成功
   return(error);
  }
//+------------------------------------------------------------------+
//| 删除一个自定义交易品种                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- 从市场报价窗口中隐藏交易品种
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- 如果删除自定义交易品种失败，在日志中报告并返回 'false'
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 把指定数量的报价读取到数组中                                     |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- 通知开始载入历史数据
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, sy

//--- 为接收报价做3次尝试
   int attempts=0;
   while(attempts<3)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 测量接收报价的开始时间
       uint start=GetTickCount();

       //--- 从 1970.01.01 00:00.001 (参数 from=1 ms) 开始请求报价历史
       int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
       if(received!=-1)
         {
          //--- 显示报价数量和花费的时间
          PrintFormat("The tick history for the '%s' symbol is received in the amount o

          //--- 如果报价历史已同步, 错误代码等于0 - 返回 'true'
          if(GetLastError()==0)
             return(true);

          PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms
                      symbol, received, GetTickCount()-start, GetLastError());
         }
       //--- 计数尝试次数
       attempts++;
       //--- 暂停一秒以等待报价数据库同步结束
       Sleep(1000);
      }
//--- 3次尝试复制报价失败
    return(false);
  }
//+------------------------------------------------------------------+
//| 返回报价的字符串类型描述                                         |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTES

//--- 检查报价标志
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)    == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)   == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)    == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)    == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)   == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- 首先检查用于交易的报价标志 (没有 CustomTicksAdd())
   if(buy_tick || sell_tick)
     {
      //--- 构建交易报价的输出内容
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tick
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tick
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
```

© 2000-2025, MetaQuotes Ltd.

```
      desc += "(Trade tick)";
     }
   else
     {
      //--- 构建信息报价的输出稍有不同
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- 返回报价描述
   return(desc);
  }
```

另见

CustomRatesDelete, CustomRatesUpdate, CustomTicksDelete, CopyTicks, CopyTicksRange

© 2000-2025, MetaQuotes Ltd.

# CustomBookAdd

为自定义交易品种传递市场深度的状态。该函数可以推广市场深度，类似于价格从交易商服务器到达。

```
bool  CustomBookAdd(
   const string         symbol,            // 交易品种名称
   const MqlBookInfo&   books[]            // 带有对市场深度元素描述的数组
   uint                 count=WHOLE_ARRAY  // 将要使用的元素数量
   );
```

参数

*交易品种*

[in] 自定义交易品种名称。

*books[]*

[in] MqlBookInfo数组类型数据充分描述了市场深度状态 —所有买卖请求。该传递的市场深度状态完全取代了过去的深度。

*count=WHOLE_ARRAY*

[in] 将要传递到函数的'books'数组元素的数量。默认使用整个数组。

返回值

true -成功，否则 -false。若要获得错误信息，请调用GetLastError()函数。

注意

CustomBookAdd函数仅适用于市场深度所开放的自定义交易品种 — 通过平台界面或MarketBookAdd函数。

当交易品种卖价和买价加入到市场深度时，不会更新。您应该控制最佳价格的变化，并使用CustomTicksAdd将其加入到报价中。

该函数验证传输数据的准确性：必须为每个元素指明类型、价格和交易量。此外，MqlBookInfo.volume和MqlBookInfo.volume_real不得为零或负数；如果两个交易量都是负数，这将被视为错误。您可以指定任何交易量或两个交易量都指定：将使用指定的交易量或正值交易量：

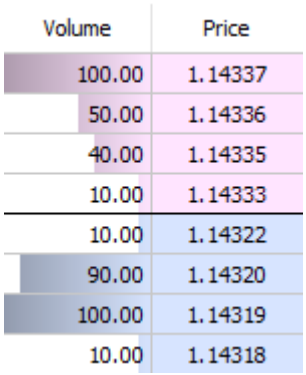
```
volume=-1 && volume_real=2 — volume_real=2将被使用，
a volume=3 && volume_real=0 — volume=3将被使用。
```

MqlBookInfo元素在'books'数组中的顺序并不重要。当保存数据时，程序端将根据其价格进行排序。

当保存数据时，检查接收人自定义交易品种的"Book depth" (SYMBOL_TICKS_BOOKDEPTH)参数。如果在传递的市场深度中，卖出请求的数量超过该值，则丢弃超出的部分。购买请求也是如此。

'books'数组的采样填充：

© 2000-2025, MetaQuotes Ltd.

| 市场深度状态 | Filling books[] |
| --- | --- |
|  | books[0].type=BOOK_TYPE_SELL;<br>books[0].price=1.14337;<br>books[0].volume=100;<br>books[1].type=BOOK_TYPE_SELL;<br>books[1].price=1.14330;<br>books[1].volume=50;<br>books[2].type=BOOK_TYPE_SELL;<br>books[2].price=1.14335;<br>books[2].volume=40;<br>books[3].type=BOOK_TYPE_SELL;<br>books[3].price=1.14333;<br>books[3].volume=10;<br>books[4].type=BOOK_TYPE_BUY;<br>books[4].price=1.14322;<br>books[4].volume=10;<br>books[5].type=BOOK_TYPE_BUY;<br>books[5].price=1.14320;<br>books[5].volume=90;<br>books[6].type=BOOK_TYPE_BUY;<br>books[6].price=1.14319;<br>books[6].volume=100;<br>books[7].type=BOOK_TYPE_BUY;<br>books[7].price=1.14318;<br>books[7].volume=10; |



例如：

```
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 为我们将要检索数据的交易品种启用市场深度 from
   MarketBookAdd(Symbol());
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
  }
//+------------------------------------------------------------------+
//| 报价函数                                                         |
//+------------------------------------------------------------------+
void OnTick(void)
  {
   MqlTick ticks[];
   ArrayResize(ticks,1);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 将普通交易品种的当前价格复制到自定义交易品种
   if(SymbolInfoTick(Symbol(),ticks[0]))
     {
      string symbol_name=Symbol()+".SYN";
      CustomTicksAdd(symbol_name,ticks);
     }
  }
//+------------------------------------------------------------------+
//| Book 函数                                                        |
//+------------------------------------------------------------------+
void OnBookEvent(const string &book_symbol)
  {
//--- 将普通交易品种的当前市场深度状态复制到自定义交易品种
   if(book_symbol==Symbol())
     {
      MqlBookInfo book_array[];
      if(MarketBookGet(Symbol(),book_array))
        {
         string symbol_name=Symbol()+".SYN";
         CustomBookAdd(symbol_name,book_array);
        }
     }
  }
//+------------------------------------------------------------------+
```

另见

MarketBookAdd, CustomTicksAdd, OnBookEvent

© 2000-2025, MetaQuotes Ltd.

# 图表操作

设置图表属性的函数(ChartSetInteger, ChartSetDouble, ChartSetString)是不同步的，用于向图表发送更新命令。如果这些函数成功执行，那么这个命令就包含在图表事件的公共队列中。图表属性的变化与该图表事件队列的处理共同实施。

因此，不要期待在调用异步函数以后立即更新图表。使用ChartRedraw()函数以强制更新图表的外观和属性。

| 函数 | 功能 |
| --- | --- |
| ChartApplyTemplate | 在图表中应用指定文件中的特定模板。 |
| ChartSaveTemplate | 以指定名称在模板中保存当前图表设置 |
| ChartWindowFind | 返回绘画指标的子窗口数量。 |
| ChartTimePriceToXY | 从表现时间/价格的图表坐标转变到X和Y轴 |
| ChartXYToTimePrice | 转变图表的X和Y轴到时间和价格值 |
| ChartOpen | 打开指定交易品种和周期的新图表。 |
| ChartClose | 关闭指定图表。 |
| ChartFirst | 返回客户端第一图表的ID。 |
| ChartNext | 返回指定图表旁边图表的ID。 |
| ChartSymbol | 返回指定图表的交易品种名称。 |
| ChartPeriod | 返回指定图表的周期值。 |
| ChartRedraw | 调用指定图表的被怕重画。 |
| ChartSetDouble | 设置指定图表相关属性的双精度值。 |
| ChartSetInteger | 设置指定图表相关属性的整数值 (日期时间，整型，颜色，布尔或者字符型)。 |
| ChartSetString | 设置指定图表相关属性的字符串值。 |
| ChartGetDouble | 返回指定图表的双精度值属性。 |
| ChartGetInteger | 返回指定图表的整数值属性。 |
| ChartGetString | 返回指定图表的字符串属性。 |
| ChartNavigate | 通过指定的关系柱完成指定图表到图表中的指定位置的转换。 |
| ChartID | 返回当前图表的ID。 |
| ChartIndicatorAdd | 用指定处理程序在指定图表窗口填加指标。 |
| ChartIndicatorDelete | 从指定图表窗口移除一个指定名称的指标。 |
| ChartIndicatorGet | 返回在指定图表窗口指定缩略名的指标句柄。 |
| ChartIndicatorName | 返回指定图表窗口的指标列表的指标缩略名数量。 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| ChartIndicatorsTotal | 返回应用到指定图表窗口全部指标数量。 |
| ChartWindowOnDropped | 返回图表子窗口EA交易或者脚本下跌到的数 (指数)。 |
| ChartPriceOnDropped | 返回图表点EA交易或者脚本下跌到的价格坐标。 |
| ChartTimeOnDropped | 返回图表点EA交易或者脚本下跌到的时间坐标。 |
| ChartXOnDropped | 返回图表点EA交易或者脚本下跌到的X坐标。 |
| ChartYOnDropped | 返回图表点EA交易或者脚本下跌到的Y坐标。 |
| ChartSetSymbolPeriod | 改变交易品种值和指定图表周期。 |
| ChartScreenShot | 以gif,png或bmp格式提供图表当前状态屏幕截图。 |

© 2000-2025, MetaQuotes Ltd.

# ChartApplyTemplate

在图表中应用指定文件中的特定模板。该命令添加到图表信息队列并且将在处理完之前所有命令后开始执行。

```
bool  ChartApplyTemplate(
   long          chart_id,      // 图表 ID
   const string  filename       // 模板文件名
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*filename*

[in] 包括模板的文件名。

返回值

如果该命令已添加到图表队列则返回true，否则false。若获得额外错误信息，需要调用函数GetLastError()。

注释

成功加载新模板时EA交易会被卸载且不能再继续操作。

当将模板应用到图表时，处于安全原因可能会限制交易权限：

> 实时交易权限不能扩展到通过使用ChartApplyTemplate()函数应用模板的EA交易。

如果调用ChartApplyTemplate()函数的mql5-程序没有交易权限，那么无论模板如何设置，通过模板启动的EA交易都将不能交易。

如果调用ChartApplyTemplate()函数的mql5-程序有交易权限，而在模板设置中没有这种权限，那么通过模板启动的EA交易将不能交易。

## 使用模板

MQL5语言资源允许使用ChartSetInteger()函数设置多个图表属性，包括颜色：

- 图表背景色；
- 坐标轴，比例和OHLC线的颜色；
- 网格颜色；
- 交易量和持仓水平的颜色；
- 牛市蜡烛图的上柱，阴影和边界的颜色；
- 熊市蜡烛图的下柱，阴影和边界的颜色；
- 图表线和Doji蜡烛图的颜色；
- 牛市蜡烛图主体的颜色；
- 熊市蜡烛图主体的颜色；
- 卖价线的颜色；

© 2000-2025, MetaQuotes Ltd.

- 买价线的颜色；
- 最后成交价线（收盘价）的颜色；
- 止损订单水平的颜色（止损和获利）。

此外，在图表上可以有多个图形对象和指标。您可以一次性以所有需要的指标设置图表，然后将其保存为模板。这种模板可以应用于任何图表。

ChartApplyTemplate() 函数意于使用之前保存的模板，并且它可以用在任何mql5程序。通过存储模板的文件路径作为ChartApplyTemplate()函数的第二参数。模板文件根据以下规则搜索：

- 如果反斜杠"\"分隔符（写为"\\"）位于路径开始，寻找 _terminal_data_directory\MQL5相对路径的模板，
- 如果没有反斜杠，则寻找调用ChartApplyTemplate()的可执行EX5文件相对的模板；
- 如果在前两个变量没有找到模板，搜索就在terminal_directory\Profiles\Templates\文件夹下执行。

这里terminal_directory是MetaTrader 5客户端运行的文件夹，terminal_data_directory是存储可编辑文件的文件夹，它的位置取决于操作系统，用户名和计算机安全设置。通常它们是不同的文件夹，但某些情况下它们可能会同时发生。

文件夹terminal_data_directory和terminal_directory的位置可以使用TerminalInfoString()函数获取。

```
//--- 启动程序端的目录
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
   Print("Terminal directory:",terminal_path);
//--- 程序端数据目录，这里存储MQL5的EA和指标文件夹
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   Print("Terminal data directory:",terminal_data_path);
```

例如：

```
//--- 在terminal_data_directory\MQL5\搜索模板
ChartApplyTemplate(0,"\\first_template.tpl"))
//--- 在directory_of_EX5_file\搜索模板，然后在文件夹terminal_data_directory\Profiles\Temp
ChartApplyTemplate(0,"second_template.tpl"))
//--- 在directory_of_EX5_file\My_templates\搜索模板，然后在文件夹terminal_directory\Profil
ChartApplyTemplate(0,"My_templates\\third_template.tpl"))
```

模板不是资源，它们不能被包含到可执行EX5文件。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 应用模板示例，位于\MQL5\Files
   if(FileIsExist("my_template.tpl"))
     {
      Print("The file my_template.tpl found in \Files'");
      //--- 应用模板
      if(ChartApplyTemplate(0,"\\Files\\my_template.tpl"))
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          Print("The template 'my_template.tpl' applied successfully");
          //--- 重绘图表
          ChartRedraw();
         }
       else
          Print("Failed to apply 'my_template.tpl', error code ",GetLastError());
     }
   else
     {
      Print("File 'my_template.tpl' not found in "
            +TerminalInfoString(TERMINAL_PATH)+"\\MQL5\\Files");
     }
  }
```

另 见

资源

© 2000-2025, MetaQuotes Ltd.

# ChartSaveTemplate

以指定名称在模板中保存当前图表设置。

```
bool  ChartSaveTemplate(
   long          chart_id,     // 图表 ID
   const string  filename      // 保存模板的文件名
   );
```

参数

*chart_id*

[in] 图表 ID。0 意味着当前图表。

*filename*

[in] 保存模板的文件名。".tpl" 扩展名将自动加入文件名；无需指定。模板被保存在 **terminal_directory**\Profiles\Templates\ 并且可以用于程序端的手动应用程序。如果同名模板已经存在，该文件的内容将被覆盖。

返回值

如果成功，函数返回true，否则返回false。若要获得有关错误的信息，请调用 GetLastError() 函数。

注意

使用模板时，您可以保存所有应用指标和图形对象的图表设置，然后将其应用到另一个图表。

例如：

```
//+------------------------------------------------------------------+
//|                                      Test_ChartSaveTemplate.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- 输入参数
input string               symbol="GBPUSD";  // 新图表的交易品种
input ENUM_TIMEFRAMES      period=PERIOD_H3; // 新图表的时间帧
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 首先附加指标到图表
   int handle;
//--- 准备使用指标
   if(!PrepareZigzag(NULL,0,handle)) return; // 失败，所以退出
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 附加指标到当前图表，但是在独立窗口。
   if(!ChartIndicatorAdd(0,1,handle))
     {
      PrintFormat("Failed to attach to chart %s/%s an indicator with the handle=%d. Ei
                  _Symbol,
                  EnumToString(_Period),
                  handle,
                  GetLastError());
      //--- 终止程序操作
      return;
     }
//--- 刷新图表看指标
   ChartRedraw();
//--- 找出最近两个Z字形断裂
   double two_values[];
   datetime two_times[];
   if(!GetLastTwoFractures(two_values,two_times,handle))
     {
      PrintFormat("Failed to find two last fractures in the Zigzag!");
      //--- 终止程序操作
      return;
     }
//--- 现在附加标准偏差通道
   string channel="StdDeviation Channel";
   if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
     {
      PrintFormat("Failed to create object %s. Error code %d",
                  EnumToString(OBJ_STDDEVCHANNEL),GetLastError());
      return;
     }
   else
     {
      //--- 通道已经创建，定义第二点
      ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
      //--- 为通道设置提示工具文本
      ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL5 Help");
      //--- 刷新图表
      ChartRedraw();
     }
//--- 在模板中保存结果
   ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- 打开新图表并应用保存的模板
   long new_chart=ChartOpen(symbol,period);
   //--- 启用图形对象的工具提示
   ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
   if(new_chart!=0)
     {
      //--- 将保存的模板应用到图表
      ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
```

© 2000-2025, MetaQuotes Ltd.

```
      }
    Sleep(10000);
   }
//+------------------------------------------------------------------+
//| 创建Z字形句柄并确保其数据准备就绪                                |
//+------------------------------------------------------------------+
bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
   {
    ResetLastError();
//--- Z字形指标必须位于terminal_data_folder\MQL5\Examples
    h=iCustom(sym,tf,"Examples\\Zigzag");
    if(h==INVALID_HANDLE)
      {
       PrintFormat("%s: Failed to create the handle of the Zigzag indicator. Error code
                   __FUNCTION__,GetLastError());
       return false;
      }
//--- 创建指标句柄时，它需要时间计算价值
    int k=0; // 等候指标计算的尝试数量
//--- 等候循环计算，如果计算还未做好准备，暂停50毫秒
    while(BarsCalculated(h)<=0)
      {
       k++;
       //--- 显示尝试的数量
       PrintFormat("%s: k=%d",__FUNCTION__,k);
       //--- 等候50毫秒，直至指标被计算
       Sleep(50);
       //--- 如果尝试超过100次，那么有些事情就是错误的
       if(k>100)
         {
          //--- 报告问题
          PrintFormat("Failed to calculate the indicator for %d attempts!");
          //--- 终止程序操作
          return false;
         }
      }
//--- 一切做好准备，指标被创建，价值被计算
    return true;
   }
//+------------------------------------------------------------------+
//| 搜索最近的2个Z字形断裂，置于数组                                 |
//+------------------------------------------------------------------+
bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int handle)
   {
    double values[];          // Z字形价值的数组
    datetime times[];         // 获得时间的数组
    int size=100;             // 数组大小
    ResetLastError();
//--- 复制指标最近的100值
```

© 2000-2025, MetaQuotes Ltd.

```
   int copied=CopyBuffer(handle,0,0,size,values);
//--- 检查复制值的数量
   if(copied<100)
     {
      PrintFormat("%s: Failed to copy %d values of the indicator with the handle=%d. E
                  __FUNCTION__,size,handle,GetLastError());
      return false;
     }
//--- 按照时间序列定义访问数组的顺序
   ArraySetAsSeries(values,true);
//--- 在这里写下断裂处的柱形数量
   int positions[];
//--- 设置数组大小
   ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positions,3);
//--- 计数器
   int i=0,k=0;
//--- 开始搜索断裂处
   while(i<100)
     {
      double v=values[i];
      //--- 我们对空值不感兴趣W
      if(v!=0.0)
        {
         //--- 记住柱形数
         positions[k]=i;
         //--- 记住断裂上的Z字形值
         get_values[k]=values[i];
         PrintFormat("%s: Zigzag[%d]=%G",__FUNCTION__,i,values[i]);
         //--- 增加计数器
         k++;
         //--- 如果找到两个断裂，中断循环
         if(k>2) break;
        }
      i++;
     }
//--- 按照时间序列定义访问数组的顺序
   ArraySetAsSeries(times,true);   ArraySetAsSeries(get_times,true);
   if(CopyTime(_Symbol,_Period,0,size,times)<=0)
     {
      PrintFormat("%s: Failed to copy %d values from CopyTime(). Error code %d",
                  __FUNCTION__,size,GetLastError());
      return false;
     }
//--- 打开柱形图的开盘时间，在这上面出现最近的2个断裂
   get_times[0]=times[positions[1]];// 倒数第二个值将被写成第一个断裂
   get_times[1]=times[positions[2]];// 倒数第三个值将是第二个断裂
   PrintFormat("%s: first=%s,  second=%s",__FUNCTION__,TimeToString(get_times[1]),Time
//--- 成功
   return true;
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另见

ChartApplyTemplate(), 资源

© 2000-2025, MetaQuotes Ltd.

# ChartWindowFind

函数返回画指标的子窗口数。该函数有两个变体。

1. 函数在指定图表中搜索带有指定的指标"缩写名"子窗口（缩写名在子窗口的左上部显示），成功的话返回子窗口数。

```
int  ChartWindowFind(
   long     chart_id,                  // 图表标识符
   string   indicator_shortname        // 短指标名，见 INDICATOR_SHORTNAME
```

2. 函数必须从自定义指标中调用。返回指标工作的子窗口数。

```
int  ChartWindowFind();
```

参量

*chart_id*

[in] 图表ID。0表示当前图表。

*indicator_shortname*

[in] 指标缩写名。

返回值

成功的话返回子窗口数。失败，函数返回-1。

注释

如果函数第二变体（无参量）是从脚本或者EA交易调用，则返回-1。

不要将指标的缩略名和文件名称弄混，这是使用iCustom()和IndicatorCreate()函数创建指标时指定的名称。如果指标的缩略名未明确设置，那么包含指标源代码的文件名称，编译期间会被指明。

正确地形成指标缩略名非常重要，它使用IndicatorSetString()函数记录在INDICATOR_SHORTNAME属性。建议缩略名包含指标输入参数的值，因为ChartIndicatorDelete()函数中从图表删除的指标通过缩略名识别。

示例：

```
#property script_show_inputs
//--- 输入参量
input string   shortname="MACD(12,26,9)";
//+------------------------------------------------------------------+
//| 返回带有这个指标的图表窗口号                                     |
//+------------------------------------------------------------------+
int GetIndicatorSubWindowNumber(long chartID=0,string short_name="")
  {
   int window=-1;
//---
   if((ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR)
     {
      //--- 函数从指标调用，名称不需要
```

© 2000-2025, MetaQuotes Ltd.

```
      window=ChartWindowFind();
     }
   else
     {
      //--- 函数从EA交易或者脚本调用
      window=ChartWindowFind(0,short_name);
      if(window==-1) Print(__FUNCTION__+"(): Error = ",GetLastError());
     }
//---
   return(window);
  }
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int window=GetIndicatorSubWindowNumber(0,shortname);
   if(window!=-1)
      Print("Indicator "+shortname+" is in the window #"+(string)window);
   else
      Print("Indicator "+shortname+" is not found. window = "+(string)window);
  }
```

另 见

ObjectCreate(), ObjectFind()

© 2000-2025, MetaQuotes Ltd.

# ChartTimePriceToXY

从表现时间/价格的图表坐标转变到X和Y轴

```
bool  ChartTimePriceToXY(
   long           chart_id,      // 图表 ID
   int            sub_window,    // 子窗口数量
   datetime       time,          // 图表上的时间轴
   double         price,         // 图表上的价格轴
   int&           x,             // 图表上时间的X轴
   int&           y              // 图表上价格的Y轴
   );
```

参数

*chart_id*

[in] 图表 ID。0意味着当前图表。

*sub_window*

[in] 图表子窗口的数量。0代表主图表窗口。

*time*

[in] 图表上的时间值，像素值将沿着X轴被接收。起点是窗口/子窗口的左上角。

*price*

[in] 图表上的价格值，像素值将沿着Y轴被接收。起点是窗口/子窗口的左上角。

*x*

[out] 变量，将接收X轴的时间转换。

*y*

[out] 变量，将接收Y轴的价格转换。

返回值

如果成功返回true，否则返回false。若要获得的错误信息，请调用GetLastError()函数。

示例：

```
#define   BAR_NUMBER    0  // 我们要读取价格和时间的柱的编号

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 根据 BAR_NUMBER 索引复制一个柱的数据
   MqlRates rates[]={};
   if(CopyRates(_Symbol, _Period, BAR_NUMBER, 1, rates)!=1)
     {
      PrintFormat("CopyRates() failed for bar %d. Error %d", BAR_NUMBER, GetLastError
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 把取得的价格和时间转换成图表的像素坐标
   int x=0, y=0;
   ResetLastError();
   if(!ChartTimePriceToXY(ChartID(), 0, rates[0].time, rates[0].close, x, y))
     {
      Print("ChartTimePriceToXY() failed. Error ", GetLastError());
      return;
     }

//--- 在日志中打印取得的结果
   PrintFormat("For bar[%d] with opening time %s and price %.*f, the chart coordinates

   /*
  结果:
   For bar[0] with opening time 2024.08.09 15:06 and price 1.27378, the chart coordina
   */
  }
```

另 见

ChartXYToTimePrice()

© 2000-2025, MetaQuotes Ltd.

# ChartXYToTimePrice

转变图表的X和Y轴到时间和价格值

```
bool  ChartXYToTimePrice(
   long            chart_id,      // 图表 ID
   int             x,             // 图表的 X 轴
   int             y,             // 图表的 Y 轴
   int&            sub_window,    // 子窗口的数量
   datetime&       time,          // 图表上的时间
   double&         price,         // 图表上的价格
   );
```

参数

*chart_id*

[in] 图表 ID。0意味着当前图表。

*x*

[in] X 轴。

*y*

[in] Y 轴。

*sub_window*

[out] 变量，将写入图表的子窗口数。0意味着主图表窗口。

*time*

[out] 图表上的时间值，像素值将沿着X轴被接收。起点是窗口/子窗口的左上角。

*price*

[out] 图表上的价格值，像素值将沿着Y轴被接收。起点是窗口/子窗口的左上角。

返回值

如果成功返回true，否则返回false。若要获得的错误信息，请调用 GetLastError() 函数。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 在图表上显示事件参数
   Comment(__FUNCTION__,": id=",id," lparam=",lparam," dparam=",dparam," sparam=",spar
//--- 如果这是鼠标点击事件图表
   if(id==CHARTEVENT_CLICK)
     {
      //--- 准备变量
      int      x     =(int)lparam;
      int      y     =(int)dparam;
      datetime dt    =0;
      double   price =0;
      int      window=0;
      //--- 依据日期/时间转换X和Y坐标
      if(ChartXYToTimePrice(0,x,y,window,dt,price))
        {
         PrintFormat("Window=%d X=%d  Y=%d  =>  Time=%s  Price=%G",window,x,y,TimeToSt
         //--- 执行反向转换:(X,Y) => (时间,价格)
         if(ChartTimePriceToXY(0,window,dt,price,x,y))
            PrintFormat("Time=%s  Price=%G  =>  X=%d  Y=%d",TimeToString(dt),price,x,y
         else
            Print("ChartTimePriceToXY return error code: ",GetLastError());
         //--- 删除线
         ObjectDelete(0,"V Line");
         ObjectDelete(0,"H Line");
         //--- 创建十字光标的水平线和垂直线
         ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
         ObjectCreate(0,"V Line",OBJ_VLINE,window,dt,price);
         ChartRedraw(0);
        }
      else
         Print("ChartXYToTimePrice return error code: ",GetLastError());
      Print("+------------------------------------------------------------------+");
     }
  }
```

See also

ChartTimePriceToXY()

© 2000-2025, MetaQuotes Ltd.

# ChartOpen

打开指定交易品种和周期的新图表。

```
long  ChartOpen(
   string           symbol,     // 交易品种名称
   ENUM_TIMEFRAMES  period      // 周期
   );
```

参量

*symbol*

[in] 图表交易品种，NULL意味着当前图表交易品种（附上EA交易）。

*period*

[in] 图表周期（时间表）。可以是ENUM_TIMEFRAMES值之一。0代表当前图表周期。

返回值

若成功，返回打开图表ID，否则返回0。

注释

程序端同时打开图表的最大值不能超过CHARTS_MAX值。

示例：

```
#define CHART_SYMBOL  NULL
#define CHART_PERIOD  PERIOD_CURRENT

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置新图表的交易品种和时间框架
   string symbol=CHART_SYMBOL;
   if(symbol==NULL)
      symbol=Symbol();
   ENUM_TIMEFRAMES timeframe = (CHART_PERIOD==PERIOD_CURRENT ? Period() : CHART_PERIOI

//--- 使用指定的交易品种和周期打开新图表
   long chart_id=ChartOpen(symbol, timeframe);
   if(chart_id==0)
     {
      Print("ChartOpen() failed. Error ", GetLastError());
      return;
     }

//--- 在日志中打印打开图表的参数
   PrintFormat("A new chart of the %s symbol has been opened with a period of %s and (
               symbol, StringSubstr(EnumToString(timeframe), 7), chart_id);
```

© 2000-2025, MetaQuotes Ltd.

```
   /*
   结果：
   A new chart of the GBPUSD symbol has been opened with a period of M1 and ChartID 1
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartFirst

返回客户端第一图表ID。

```
long  ChartFirst();
```

返回值

图表 ID。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得客户端中第一个打开的图表的ID
   long            chart_id= ChartFirst();

//--- 再使用获取的ID取得图表的交易品种和周期
   string          symbol  = ChartSymbol(chart_id);
   ENUM_TIMEFRAMES period  = ChartPeriod(chart_id);

//--- 在日志中显示客户端中第一个图表的描述
   PrintFormat("ID of the first chart of the client terminal: %I64d, chart symbol: %s,
   /*
   结果：
   ID of the first chart of the client terminal: 133246248352168440, chart symbol: EUR
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartNext

返回指定图表旁边的图表ID。

```
long  ChartNext(
   long  chart_id      // 图表 ID
   );
```

参量

*chart_id*

[in] 图表 ID. 0 不表示当前图表。0表示 "返回第一图表"。

返回值

图表 ID。如果在图表列表末端，返回-1 。

示例：

```
//--- 用于图表 ID的变量
   long currChart,prevChart=ChartFirst();
   int i=0,limit=100;
   Print("ChartFirst =",ChartSymbol(prevChart)," ID =",prevChart);
   while(i<limit)// 不允许超过100个打开的窗口
     {
      currChart=ChartNext(prevChart); // 通过使用之前图表ID获得新图表ID
      if(currChart<0) break;          // 到达了图表列表末端
      Print(i,ChartSymbol(currChart)," ID =",currChart);
      prevChart=currChart;// 为ChartNext()保存当前图表ID
      i++;// 不要忘记增加计数器
     }
```

© 2000-2025, MetaQuotes Ltd.

# ChartClose

关闭指定图表。

```
bool  ChartClose(
   long  chart_id=0      // 图表 ID
   );
```

参量

*chart_id=0*

[in] 图表 ID. 0 意味着当前图表。

返回值

若成功，返回true，否则false。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打开一个与当前图表具有相同交易品种和周期的新图表
   long chart_id=ChartOpen(_Symbol, _Period);
   if(chart_id==0)
     {
      Print("ChartOpen() failed. Error ", GetLastError());
      return;
     }

//--- 在日志中打印打开图表的参数
   PrintFormat("A new chart of the %s symbol has been opened with a period of %s and C
               _Symbol, StringSubstr(EnumToString(_Period), 7), chart_id);

//--- 等待三秒再关闭新打开的图表
   PrintFormat("Waiting 3 seconds before closing a newly opened chart with ID %I64d ..
   Sleep(3000);
   ResetLastError();
   if(!ChartClose(chart_id))
     {
      Print("ChartClose() failed. Error ", GetLastError());
      return;
     }

//--- 在日志中打印成功关闭图表的消息
   PrintFormat("The chart with ID %I64d was successfully closed", chart_id);
   /*
   结果：
   A new chart of the GBPUSD symbol has been opened with a period of M1 and ChartID 13
```

© 2000-2025, MetaQuotes Ltd.

```
   Waiting 3 seconds before closing a newly opened chart with ID 133346697706632016 ..
   The chart with ID 133346697706632016 was successfully closed
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartSymbol

返回指定图表的交易品种名称。

```
string  ChartSymbol(
   long  chart_id=0      // 图表 ID
   );
```

参量

*chart_id=0*

[in] 图表 ID. 0 意味着当前图表。

返回值

如果图表不存在，结果是空字符串。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表的交易品种并在日志中显示所取得的值
   string chart_symbol = ChartSymbol();
   Print("Current chart symbol: ", chart_symbol);

//--- 获取已有(本例中为当前)图表的ID
   long chart_id=ChartID();
   chart_symbol=ChartSymbol(chart_id);
   PrintFormat("Chart symbol with ID %I64d: %s", chart_id, chart_symbol);

//--- 当得到交易品种时设置随机图表ID
   chart_symbol = ChartSymbol(1234567890);
   if(chart_symbol=="")
      Print("The chart with ID 1234567890 does not exist");
   else
      Print("Chart symbol with ID 1234567890: ", chart_symbol);
   /*
   结果:
   Current chart symbol: GBPUSD
   Chart symbol with ID 132966427583395104: GBPUSD
   The chart with ID 1234567890 does not exist
   */
  }
```

另见

ChartSetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

# ChartPeriod

返回指定图表的时间表 周期 。

```
ENUM_TIMEFRAMES  ChartPeriod(
   long  chart_id=0      // 图表 ID
   );
```

参量

*chart_id=0*

[in] 图表 ID. 0 意味着当前图表。

返回值

函数返回ENUM_TIMEFRAMES值之一，如果图表不存在，返回0。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表的周期并在日志中显示取得的值
   ENUM_TIMEFRAMES period=ChartPeriod();
   Print("Current chart period: ", EnumToString(period));

//--- 获取已有(本例中为当前)图表的ID
   long chart_id=ChartID();
   period=ChartPeriod(chart_id);
   PrintFormat("Chart period with ID %I64d: %s", chart_id, EnumToString(period));

//--- 设置一个随机的图表ID
   period=ChartPeriod(1234567890);
   if(period==0)
      Print("The chart with ID 1234567890 does not exist");
   else
      Print("Chart period with ID 1234567890: ", EnumToString(period));
   /*
   结果:
   Current chart period: PERIOD_M15
   Chart period with ID 133510090240498505: PERIOD_M15
   The chart with ID 1234567890 does not exist
   */
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartRedraw

指定图表被迫重画调用此函数。

```
void  ChartRedraw(
   long  chart_id=0      // 图表 ID
   );
```

参量

*chart_id=0*

[in] 图表 ID. 0 意味着当前图表。.

注释

通常更改物件属性之后使用该函数。

示例：

```
#define   WIDTH      50    // 以柱数表示的矩形宽度

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表ID并设置图形对象名称
   long   chart_id = ChartID();
   string obj_name = MQLInfoString(MQL_PROGRAM_NAME)+"_RectLabel";

//--- 取得两个矩形的时间坐标
   datetime time1  = iTime(_Symbol, _Period, WIDTH);
   datetime time2  = iTime(_Symbol, _Period, 0);
   if(time1==0 || time2==0)
     {
      Print("Error getting time ", GetLastError());
      return;
     }

//--- 取得矩形宽度范围内的最高和最低价格
   double price1 = HighestHigh(_Symbol, _Period, 0, WIDTH);
   double price2 = LowestLow(_Symbol, _Period, 0, WIDTH);
   if(price1==EMPTY_VALUE || price2==EMPTY_VALUE)
      return;

//--- 创建一个矩形对象
   Print("Create a wheat-colored rectangle");
   if(!ObjectCreate(chart_id, obj_name, OBJ_RECTANGLE, 0, time1, price1, time2, price2
     {
      Print("ObjectCreate() failed. Error ", GetLastError());
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 使用原始颜色填充矩形
   ObjectSetInteger(chart_id, obj_name, OBJPROP_FILL, true);
   ObjectSetInteger(chart_id, obj_name, OBJPROP_BACK, true);
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrWheat);
   ChartRedraw();

//--- 等待一秒钟, 使用道奇蓝色填充矩形并更新图表
   Sleep(1000);
   Print("Change color to DodgerBlue");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrDodgerBlue);
   ChartRedraw();

//--- 等待一秒钟, 使用酸橙绿色填充矩形并更新图表
   Sleep(1000);
   Print("Change color to LimeGreen");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrLimeGreen);
   ChartRedraw();

//--- 等待一秒钟, 使用橙红色填充矩形并更新图表
   Sleep(1000);
   Print("Change color to OrangeRed");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrOrangeRed);
   ChartRedraw();

//--- 等待一秒钟, 使用小麦色填充矩形并更新图表
   Sleep(1000);
   Print("Reset color to original");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrWheat);
   ChartRedraw();

//--- 三秒钟后, 从图表上删除对象
   Sleep(3000);
   Print("Delete the rectangle");
   ObjectDelete(chart_id, obj_name);
  }
//+------------------------------------------------------------------+
//| 返回指定柱数范围内的最高价格                                     |
//+------------------------------------------------------------------+
double HighestHigh(const string symbol, const ENUM_TIMEFRAMES timeframe, const uint st
  {
   ResetLastError();
   int index=iHighest(symbol, timeframe, MODE_HIGH, count, start);
   if(index==-1)
     {
      PrintFormat("%s: iHighest() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   GetLastError();
   double price=iHigh(symbol, timeframe, index);
   if(price==0)
     {
      PrintFormat("%s: iHigh() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   return(price);
  }
//+------------------------------------------------------------------+
//| 返回指定柱数范围内的最低价格                                     |
//+------------------------------------------------------------------+
double LowestLow(const string symbol, const ENUM_TIMEFRAMES timeframe, const uint sta
  {
   ResetLastError();
   int index=iLowest(symbol, timeframe, MODE_LOW, count, start);
   if(index==-1)
     {
      PrintFormat("%s: iLowest() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   GetLastError();
   double price=iLow(symbol, timeframe, index);
   if(price==0)
     {
      PrintFormat("%s: iLow() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   return(price);
  }
```

另见

对象函数

© 2000-2025, MetaQuotes Ltd.

# ChartSetDouble

设置指定图表相关属性值。图表属性是双精度类型。该命令添加到图表信息队列并且将在处理完之前所有命令后开始执行。

```
bool  ChartSetDouble(
   long                            chart_id,      // 图表 ID
   ENUM_CHART_PROPERTY_DOUBLE      prop_id,       // 属性 ID
   double                          value          // 值
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*prop_id*

[in] 图表属性 ID. 可以是 ENUM_CHART_PROPERTY_DOUBLE值中的一个 (除了只读属性)。

*value*

[in] 属性值。

返回值

如果该命令已添加到图表队列则返回true，否则false。若获得额外错误信息，需要调用函数 GetLastError()。

注意

该函数为异步函数，这意味着该函数不等待执行已成功添加到指定图表队列的命令。相反，而是立即返回控制。该属性将只会在处理图表队列的对应命令之后更改。若要立即执行来自图表队列的命令，请调用ChartRedraw函数。

如果您想立即改变多个图表属性，那么应该在一个代码框中执行对应的函数 (ChartSetString, ChartSetDouble, ChartSetString)，在此之后您应该马上调用ChartRedraw。

若要检查命令执行的结果，您可以使用一个函数，请求指定的图表属性 (ChartGetInteger, ChartGetDouble, ChartSetString)。然而，请注意，这些都是同步函数，等待执行结果。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表ID
   long chart_id = ChartID();

//--- 保存从右边数第零柱的初始偏移百分比和自动滚动标志
   double shift = ChartGetDouble(chart_id, CHART_SHIFT_SIZE);
   bool   scroll= ChartGetInteger(chart_id,CHART_AUTOSCROLL);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 重置自动滚动图表标志
   ChartSetInteger(chart_id, CHART_AUTOSCROLL, true);
   PrintFormat("Initial chart shift size: %.2f", shift);

//--- 从2.0%到50.0%以步长0.5%做循环
   for(int i=20; i<=500; i+=5)
     {
      //--- 以平移大小为 i / 10 并在日志中打印指定的数值
      ChartSetDouble(chart_id, CHART_SHIFT_SIZE, i/10.0);
      PrintFormat("Set chart shift size to %.1f%%", i/10.0);

      //--- 使图表的第零柱保持在指定的偏移距离
      ChartNavigate(chart_id, CHART_END, 0);

      //--- 等待一会儿
      Sleep(16);
     }

//--- 设置初始的图表偏移和自动滚动
   Print("Set the chart shift size to the initial value of ", shift);
   ChartSetDouble(ChartID(), CHART_SHIFT_SIZE, shift);
   ChartSetInteger(chart_id, CHART_AUTOSCROLL, scroll);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartSetInteger

设置指定图表相关属性值。图表属性必须是日期时间，整型，颜色，布尔型或者字符型。该命令添加到图表信息队列并且将在处理完之前所有命令后开始执行。

```
bool  ChartSetInteger(
   long                          chart_id,      // 图表 ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,       // 属性 ID
   long                          value          // 值
   );
```

```
bool  ChartSetInteger(
   long                          chart_id,      // 图表 ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,       // 属性 ID
   int                           sub_window,    // 子窗口号
   long                          value          // 值
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*prop_id*

[in] 图表属性ID. 可以是ENUM_CHART_PROPERTY_INTEGER值中的一个 (除了只读属性)。

*sub_window*

[in] 图表子窗口数。对于第一种情况，默认值为0 (图表主窗口)。属性大部分情况不需要子窗口数。

*值*

[in] 属性值。

返回值

如果该命令已添加到图表队列则返回true，否则false。若获得额外错误信息，需要调用函数GetLastError()。

注意

该函数为异步函数，这意味着该函数不等待执行已成功添加到指定图表队列的命令。相反，而是立即返回控制。该属性将只会在处理图表队列的对应命令之后更改。若要立即执行来自图表队列的命令，请调用ChartRedraw函数。

如果您想立即改变多个图表属性，那么应该在一个代码框中执行对应的函数 (ChartSetString, ChartSetDouble, ChartSetString)，在此之后您应该马上调用ChartRedraw。

若要检查命令执行的结果，您可以使用一个函数，请求指定的图表属性 (ChartGetInteger, ChartGetDouble, ChartSetString)。然而，请注意，这些都是同步函数，等待执行结果。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 在图表窗口中启用鼠标移动的事件
   ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
//--- 强制更新图表属性确保做好事件处理的准备
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| 鼠标点选位置                                                     |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // 鼠标左移
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // 鼠标右移
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // 鼠标居中
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // 鼠标点选第一个X键
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // 鼠标点选第二个 X 键
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // 移位键
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // 控制键
   return(res);
  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
  {
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartSetString

设置指定图表的相关属性值。图表属性必须是字符串类型。该命令添加到图表信息队列并且将在处理完之前所有命令后开始执行。

```
bool  ChartSetString(
   long                          chart_id,    // 图表 ID
   ENUM_CHART_PROPERTY_STRING    prop_id,     // 属性 ID
   string                        str_value    // 值
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*prop_id*

[in] 图表属性 ID. 可以是ENUM_CHART_PROPERTY_STRING值中的一个（除了只读属性）。

*str_value*

[in] 属性值字符串。字符串长度不能超过 2045 个字符（多出的字符会被删除）。

返回值

如果该命令已添加到图表队列则返回true，否则false。若获得额外错误信息，需要调用函数GetLastError()。

注释

ChartSetString可以代替注释函数用作图标中的注释输出。

该函数为异步函数，这意味着该函数不等待执行已成功添加到指定图表队列的命令。相反，而是立即返回控制。该属性将只会在处理图表队列的对应命令之后更改。若要立即执行来自图表队列的命令，请调用ChartRedraw函数。

如果您想立即改变多个图表属性，那么应该在一个代码框中执行对应的函数（ChartSetString, ChartSetDouble, ChartSetString），在此之后您应该马上调用ChartRedraw。

若要检查命令执行的结果，您可以使用一个函数，请求指定的图表属性（ChartGetInteger, ChartGetDouble, ChartSetString）。然而，请注意，这些都是同步函数，等待执行结果。

示例：

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
   string comment=StringFormat("Выводим цены:\nAsk = %G\nBid = %G\nSpread = %d",
                               Ask,Bid,Spread);
   ChartSetString(0,CHART_COMMENT,comment);
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另见

注释， ChartGetString

© 2000-2025, MetaQuotes Ltd.

# ChartGetDouble

返回指定图表的相关属性值。图表属性必须是双精度类型。有两个函数调用变体。

1. 直接返回属性值。

```
double  ChartGetDouble(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,          // 属性 ID
   int                           sub_window=0      // 子窗口号，如果需要的话
   );
```

2. 根据函数成功与否，返回 true 或 false。如果成功，属性值置于引用传递的目标变量double_var。

```
bool  ChartGetDouble(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,          // 属性 ID
   int                           sub_window,       // 子窗口号
   double&                       double_var        // 用于图表属性的目标变量
   );
```

参量

*chart_id*

[in] 图表 ID。0 意味着当前图表。

*prop_id*

[in] 图表属性 ID。该值可以是ENUM_CHART_PROPERTY_DOUBLE值中之一。

*sub_window*

[in] 图表子窗口数。对于第一种情况，默认值为0（图表主窗口）。属性大部分情况不需要子窗口数。

*double_var*

[out] 需求属性双精度类型的目标变量。

返回值

双精度值。

对于第二种调用情况，如果指定属性有效且其值置于double_var变量，返回true，否则返回false。若获得额外错误信息，需要调用函数GetLastError()。

注意

该函数为同步函数，这意味着它等待执行在其调用之前已添加到图表队列的所有命令。

示例：

```
void OnStart()
  {
   double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,0);
   double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,0);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("CHART_PRICE_MIN =",priceMin);
   Print("CHART_PRICE_MAX =",priceMax);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartGetInteger

返回指定图表相关属性值。图表属性必须是日期时间，整型，布尔型类型。有两种函数调用变体。

1. 直接返回属性值。

```
long  ChartGetInteger(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_INTEGER   prop_id,          // 属性 ID
   int                           sub_window=0      // 子窗口号，如果需要的话
   );
```

2. 按照函数成功与否，返回 true 或者 false。如果成功，属性值置于引用传递的目标变量long_var。

```
bool  ChartGetInteger(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_INTEGER   prop_id,          // 属性 ID
   int                           sub_window,       // 子窗口号
   long&                         long_var          // 属性目标值
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*prop_id*

[in] 图表属性 ID. 该值可以是 ENUM_CHART_PROPERTY_INTEGER 值中的一个。

*sub_window*

[in] 图表子窗口数。对于第一种情况，默认值为0 (图表主窗口)。属性大部分情况都不需要子窗口数。

*long_var*

[out] 请求属性的长整型目标变量。

返回值

长整型值。

对于第二种情况，如果指定属性有效并且其值存储在long_var变量中，则返回true，否则返回false。若获得额外错误信息，需要调用 GetLastError()函数。

注意

该函数为同步函数，这意味着它等待执行在其调用之前已添加到图表队列的所有命令。

示例：

```
void OnStart()
  {
   int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
```

© 2000-2025, MetaQuotes Ltd.

```
   int width=ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   Print("CHART_HEIGHT_IN_PIXELS =",height,"pixels");
   Print("CHART_WIDTH_IN_PIXELS =",width,"pixels");
  }
```

© 2000-2025, MetaQuotes Ltd.

# ChartGetString

返回指定图表的相关属性值。图表属性必须是字符串型。有两个函数调用变体。

1. 直接返回属性值。

```
string  ChartGetString(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_STRING    prop_id           // 属性 ID
   );
```

2. 按照函数成功与否，返回true或者false。如果成功，属性值置于引用传递的目标变量string_var。

```
bool  ChartGetString(
   long                          chart_id,         // 图表 ID
   ENUM_CHART_PROPERTY_STRING    prop_id,          // 属性 ID
   string&                       string_var        // 属性目标值
   );
```

参数

*chart_id*

[in] 图表 ID。0 意味着当前图表。

*prop_id*

[in] 图表属性 ID。该值可以是 ENUM_CHART_PROPERTY_STRING 值中的一个。

*string_var*

[out] 请求属性字符串类型的目标变量。

返回值

字符串类型值。

对于第二种调用情况，如果指定属性有效并且其值存储在string_var变量中返回true，否则返回 false。若要获得额外错误信息，需要调用GetLastError()函数。

注释

ChartGetString 可以使用注释或者ChartSetString函数用来读取图表中标绘的注释。

该函数为同步函数，这意味着它等待执行在其调用之前已添加到图表队列的所有命令。

示例：

```
void OnStart()
  {
   ChartSetString(0,CHART_COMMENT,"Test comment.\nSecond line.\nThird!");
   ChartRedraw();
   Sleep(1000);
   string comm=ChartGetString(0,CHART_COMMENT);
   Print(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

另见

注释，ChartSetString

© 2000-2025, MetaQuotes Ltd.

# ChartNavigate

通过指定的关系柱完成指定图表到图表中的指定位置的转换。

```
bool  ChartNavigate(
   long                  chart_id,      // 图表 ID
   ENUM_CHART_POSITION   position,      // 位置
   int                   shift=0        // 移动值
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*position*

[in] 完成转换的图表位置。可以是ENUM_CHART_POSITION值中的一个。

*shift=0*

[in] 转换图表的柱数。正值意味着向右移动 (图表末端)，负值意味着左移 (到图表起点)。0值用来控制图表的起止。

返回值

若成功，返回true，否则返回false。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 获得当前图表的句柄
   long handle=ChartID();
   string comm="";
   if(handle>0) // 如果成功，另外设置图表
     {
      //--- 禁止自动滚动
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- 设置从右图表边界转移
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- 绘制蜡烛图
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- 设置跳动量的显示模式
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

      //--- 在Comment()准备输出文本
      comm="Scroll 10 bars to the right of the history start";
      //--- 显示注释
      Comment(comm);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 滚动10柱到历史记录开始的右侧
      ChartNavigate(handle,CHART_BEGIN,10);
      //--- 设置图表上第一柱显示的数量（类似时间帧的编号）
      long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      //--- 添加换行符
      comm=comm+"\r\n";
      //--- 添加到注释
      comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\r
      //--- 显示注释
      Comment(comm);
      //--- 等待5秒查看图表如何移动
      Sleep(5000);

      //--- 添加到注释文本
      comm=comm+"\r\n"+"Scroll 10 bars to the left of the right chart border";
      Comment(comm);
      //--- 滚动10柱到右图表边界的左侧
      ChartNavigate(handle,CHART_END,-10);
      //--- 获得图表上第一注显示的数量（类似时间帧的编号）
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\r
      Comment(comm);
      //--- 等待5秒查看图表如何移动
      Sleep(5000);

      //--- 图表滚动的新模块
      comm=comm+"\r\n"+"Scroll 300 bars to the right of the history start";
      Comment(comm);
      //--- 滚动300柱到历史记录开始的右侧
      ChartNavigate(handle,CHART_BEGIN,300);
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\r
      Comment(comm);
      //--- 等待5秒查看图表如何移动
      Sleep(5000);

      //--- 图表滚动的新模块
      comm=comm+"\r\n"+"Scroll 300 bars to the left of the right chart border";
      Comment(comm);
      //--- 滚动300柱到右图表边界的左侧
      ChartNavigate(handle,CHART_END,-300);
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\r
      Comment(comm);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# ChartID

返回当前图表ID。

```
long  ChartID();
```

返回值

长整型 值。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 用于图表ID的变量
   long curr_chart=ChartFirst();
   int i=0;
   //--- 在日志中打印第一个图表的数据
   PrintFormat("Chart[%d] ID: %I64d,  symbol: %s", i, curr_chart, ChartSymbol(curr_cha

//--- 直到达到打开图表的限制(CHARTS_MAX)
   while(!IsStopped() && i < CHARTS_MAX)
     {
      //--- 递增图表计数器
      i++;
      //--- 基于前一个ID取得下一个图表的ID
      curr_chart=ChartNext(curr_chart);

      //--- 如果到达图表列表末尾就终止循环
      if(curr_chart<0)
         break;

      //--- 在日志中打印下一个图表的数据
      PrintFormat("Chart[%d] ID: %I64d,  symbol: %s", i, curr_chart, ChartSymbol(curr_
     }
   /*
  结果：
   Chart[0] ID: 133246248352168440,  symbol: EURUSD
   Chart[1] ID: 133346697706632015,  symbol: USDJPY
   Chart[2] ID: 133246248352168439,  symbol: GBPUSD
   Chart[3] ID: 133346697706632009,  symbol: RU000A103661
   Chart[4] ID: 133346697706632010,  symbol: AEM4
   Chart[5] ID: 133346697706632011,  symbol: AA.SPB
   Chart[6] ID: 133346697706632012,  symbol: ALLFUTMIX
   Chart[7] ID: 133346697706632013,  symbol: EURUSD
   Chart[8] ID: 133346697706632014,  symbol: SBER
   */
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# ChartIndicatorAdd

用指定处理程序在指定图表窗口填加指标。Indicator and chart should be generated on the same symbol and time frame.

```
bool  ChartIndicatorAdd(
   long  chart_id,                  // 图表 ID
   int   sub_window                 // 子窗口号
   int   indicator_handle           // 指标处理
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*sub_window*

[in] 图表子窗口数。0代表图表主窗口。若要在新窗口添加指标，参数必须比现有最近窗口的标引大一，例如等于CHART_WINDOWS_TOTAL。如果参数值大于CHART_WINDOWS_TOTAL，新窗口将不会创建，指标也不会添加。

*indicator_handle*

[in] 指标处理程序。

返回值

若成功函数返回true，否则返回false。若要获得额外错误信息，需要调用GetLastError()函数。Error 4114 means that a chart and an added indicator differ by their symbol or time frame.

注意

如果应该在独立子窗口绘制的指标（例如，内置iMACD或指定#property indicator_separate_window属性的自定义指标被应用到图表主窗口，那么虽然它仍然显示在指标列表，却还是可能看不见。这意味着指标比例不同于价格图表比例，应用的指标值不适用于价格图表的显示范围。这种情况下，GetLastError()返回0代码表示没有错误。这种“不可见”的指标值可以在数据窗口看见并且从其他MQL5应用程序接收。

另见

ChartIndicatorDelete(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate()

© 2000-2025, MetaQuotes Ltd.

# ChartIndicatorDelete

从指定图表窗口删除指定名称的指标。

```
bool  ChartIndicatorDelete(
   long           chart_id,              // 图表 id
   int            sub_window             // 子窗口数量
   const string   indicator_shortname    // 指标的缩略名
   );
```

参数

*chart_id*

[in] 图表 ID。0 表示当前图表。

*sub_window*

[in] 图表子窗口的数量。0代表主图表的子窗口。

*const indicator_shortname*

[in] 通过IndicatorSetString()函数在INDICATOR_SHORTNAME属性设置的指标的缩略名。若要获得指标的缩略名，请使用ChartIndicatorName()函数。

返回值

如果指标成功删除，返回true。否则返回false。若要获得错误信息，请使用GetLastError()函数。

注意

如果相同缩略名的两个指标都在图表子窗口，这行的第一个将被删除。

如果该图表上的另一个指标基于正被删除的指标值，这种指标也将被删除。

不要混淆指标缩略名和使用函数iCustom()和IndicatorCreate()创建指标时指定的文件名。如果指标缩略名没有明确设置，那么包含指标源代码的文件名将在编译期被指明。

删除来自图表的指标不表示其计算部分将被从程序端内存删除。若要释放指标句柄，请使用IndicatorRelease()函数。

指标缩略名应该正确设置。它将使用IndicatorSetString()函数被写入INDICATOR_SHORTNAME属性。建议缩略名应该包含指标所有输入参数的值，因为通过ChartIndicatorDelete()函数从图表删除的指标通过缩略名识别。

初始化失败后删除指标的示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                    Demo_ChartIndicatorDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 绘制柱状图
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      first_param=1;
input int      second_param=2;
input int      third_param=3;
input bool     wrong_init=true;
//--- 指标缓冲区
double         HistogramBuffer[];
string         shortname;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
   int res=INIT_SUCCEEDED;
//--- 连接柱状图缓冲区数组到指标缓冲区
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- 基于输入参数设置指标缩略名
   shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                          first_param,second_param,third_param);
   IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- 如果设置指标强制完成，返回非零值
   if(wrong_init) res=INIT_FAILED;
   return(res);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 工作循环的起始位置
   int start=prev_calculated-1;
   if(start<0) start=0;
//--- 填写指标缓冲区的值
   for(int i=start;i<rates_total;i++)
     {
```

```
      HistogramBuffer[i]=close[i];
     }
//--- 返回prev_calculated值为了下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Deinit 事件的处理程序                                            |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   PrintFormat("%s: Deinitialization reason code=%d",__FUNCTION__,reason);
   if(reason==REASON_INITFAILED)
     {
      PrintFormat("An indicator with a short name %s (file %s) deletes itself from the
      int window=ChartWindowFind();
      bool res=ChartIndicatorDelete(0,window,shortname);
      //--- 分析调用ChartIndicatorDelete()的结果
      if(!res)
        {
         PrintFormat("Failed to delete indicator %s from window #%d. Error code %d",
                     shortname,window,GetLastError());
        }
     }
  }
```

另 见

ChartIndicatorAdd(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

© 2000-2025, MetaQuotes Ltd.

# ChartIndicatorGet

在指定图表窗口返回指定缩略名的指标句柄。

```
int  ChartIndicatorGet(
   long           chart_id,              // 图表 ID
   int            sub_window             // 子窗口数量
   const string   indicator_shortname    // 指标的缩略名
   );
```

参数

*chart_id*

[in] 图表 ID。0 表示当前图表。

*sub_window*

[in] 图表子窗口的数量。0代表图表主窗口。

*const indicator_shortname*

[in] 使用IndicatorSetString() 函数在INDICATOR_SHORTNAME 属性设置的指标的缩略名。若要获得指标的缩略名，请使用ChartIndicatorName()函数。

返回值

如果成功返回指标句柄，否则返回 INVALID_HANDLE。 所有获得错误的信息，请调用 GetLastError() 。

注意

ChartIndicatorGet()関数を使用して取得した指標ハンドルは、内部指標使用カウンターを増やします。カウンターがゼロより大きいすべての指標は、ターミナルランタイムシステムに読み込まれたままになります。したがって、以下の例に示すように、不要になった指標ハンドルは、IndicatorRelease()を使用して、それを受け取ったのと同じプログラムですぐに明示的に解放する必要があります。別のプログラムから「放棄された」ハンドルを見つけて正しく解放することはできません。

创建指标时，谨慎设置缩略名，它将使用IndicatorSetString()函数被写入 INDICATOR_SHORTNAME属性。建议缩略名应该包含指标所有输入参数的 值，因为在 ChartIndicatorGet()函数指标基于缩略名识别。

识别指标的另一种方式就是使用IndicatorParameters()函数获取给定句柄的参数列表，然后分析获得的值。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- 图表上的窗口数量(至少一个主窗口始终存在)
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   //--- 检查所有窗口
   for(int w=0;w<windows;w++)
     {
      //--- 该窗口/子窗口的指标数量
      int total=ChartIndicatorsTotal(0,w);
      //--- 检查窗口中的所有指标
      for(int i=0;i<total;i++)
        {
         //--- 获得指标的缩略名
         string name=ChartIndicatorName(0,w,i);
         //--- 获得指标的句柄
         int handle=ChartIndicatorGet(0,w,name);
         //--- 添加到日志
         PrintFormat("Window=%d,  index=%d,  Name=%s,  handle=%d",w,i,name,handle);
         //--- 当不再需要时您应该强制释放指标句柄
         IndicatorRelease(handle);
        }
     }
  }
```

另 见

ChartIndicatorAdd(), ChartIndicatorName(), ChartIndicatorsTotal(), IndicatorParameters()

© 2000-2025, MetaQuotes Ltd.

# ChartIndicatorName

在指定图表窗口根据指标列表编号返回指标缩略名。

```
string  ChartIndicatorName(
   long   chart_id,       // 图表 id
   int    sub_window      // 子窗口数量
   int    index           // 添加到图表子窗口的指标列表中的指标标引
   );
```

参数

*chart_id*

[in] 图表 ID。0 表示当前图表。

*sub_window*

[in] 图表子窗口的数量。0代表主图表的子窗口。

*index*

[in] 指标列表中的指标标引。指标编号从0开始，例如列表中的第一个指标编号为0。若要获得列表中的指标数量，请使用ChartIndicatorsTotal()函数。

返回值

使用IndicatorSetString() 函数在INDICATOR_SHORTNAME 属性设置的指标的缩略名。若要获得错误信息，请使用 GetLastError() 函数。

注意

不要混淆指标缩略名和使用函数 iCustom() 和 IndicatorCreate()创建指标时指定的文件名。如果指标缩略名没有明确设置，那么包含指标源代码的文件名将在编译期被指明。

指标缩略名应该正确设置。它将使用IndicatorSetString() 函数被写入INDICATOR_SHORTNAME 属性。建议缩略名应该包含指标所有输入参数的值，因为通过ChartIndicatorDelete() 函数从图表删除的指标通过缩略名识别。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表ID和它的主窗口加子窗口的数量
   long chart_id = ChartID();
   int  wnd_total= (int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);

//--- 在当前图表的所有窗口中循环迭代
   for(int w=0; w<wnd_total; w++)
     {
      //--- 根据指定的循环索引取得附加到图表窗口中的指标数量
      int ind_total=ChartIndicatorsTotal(chart_id, w);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 打印所选图表窗口的标题
      PrintFormat("Chart window %d indicators: ", w);

      //--- 在循环中写入附加到所选窗口的所有指标的名称
      string ind_names="";
      for(int i=0; i<ind_total; i++)
        {
         ind_names+="  "+ChartIndicatorName(chart_id, w, i)+(i<ind_total-1 ? "\n": "")
        }
      //--- 在日志中打印附加于指定图表窗口的所有指标的名称
      Print(ind_names);
     }
   /*
   结果:
   Chart window 0 indicators:
     AMA(9,2,30)
     SAR(0.02,0.20)
     Fractals
   Chart window 1 indicators:
     RSI(14)
     AMA(9,2,30)
   Chart window 2 indicators:
     MFI(14)
   */
  }
```

另 见

ChartIndicatorAdd(), ChartIndicatorDelete(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

© 2000-2025, MetaQuotes Ltd.

# ChartIndicatorsTotal

返回应用到指定图表窗口的指标数量。

```
int  ChartIndicatorsTotal(
   long  chart_id,      // 图表 id
   int   sub_window     // 子窗口数量
   );
```

参数

*chart_id*

[in] 图表ID。0表示当前图表。

*sub_window*

[in] 图表子窗口的数量。0代表主图表的子窗口。

返回值

指定图表窗口的指标数量。若要获得错误信息，请使用 GetLastError() 函数。

注意

函数允许搜寻附加在图表的所有指标。图表窗口的总数可以使用ChartGetInteger()函数从CHART_WINDOWS_TOTAL属性获得。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表ID和它的主窗口加子窗口的数量
   long chart_id = ChartID();
   int  wnd_total= (int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);

//--- 在当前图表的所有窗口中循环迭代
   for(int w=0; w<wnd_total; w++)
     {
      //--- 根据指定的循环索引取得附加到图表窗口中的指标数量
      int ind_total=ChartIndicatorsTotal(chart_id, w);

      //--- 取得附加于指定图表窗口的指标数量
      PrintFormat("Chart ID %I64d, subwindow %d. Attached indicators: %d", chart_id, w
     }
   /*
   结果:
   Chart ID 133246248352168439, subwindow 0. Attached indicators: 3
   Chart ID 133246248352168439, subwindow 1. Attached indicators: 2
   Chart ID 133246248352168439, subwindow 2. Attached indicators: 1
   */
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另 见

ChartIndicatorAdd(), ChartIndicatorDelete(), iCustom(), IndicatorCreate(), IndicatorSetString()

© 2000-2025, MetaQuotes Ltd.

# ChartWindowOnDropped

返回图表子窗口EA交易或者脚本下跌到的数 (指数)。0代表图表主窗口。

```
int  ChartWindowOnDropped();
```

返回值

整型 值。

示例：

```
   int myWindow=ChartWindowOnDropped();
   int windowsTotal=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   Print("Script is running on the window #"+myWindow+
         ". Total windows on the chart "+ChartSymbol()+":",windowsTotal);
```

另见

ChartPriceOnDropped, ChartTimeOnDropped, ChartXOnDropped, ChartYOnDropped

© 2000-2025, MetaQuotes Ltd.

# ChartPriceOnDropped

返回图表点一致的EA交易或者脚本下跌到的价格坐标。

```
double  ChartPriceOnDropped();
```

返回值

双精度值

示例：

```
   double p=ChartPriceOnDropped();
   Print("ChartPriceOnDropped() = ",p);
```

另见

ChartXOnDropped, ChartYOnDropped

© 2000-2025, MetaQuotes Ltd.

# ChartTimeOnDropped

返回图表点一致的EA交易或者脚本下跌到的时间坐标。

```
datetime  ChartTimeOnDropped();
```

返回值

日期时间 类型值。

示例：

```
   datetime t=ChartTimeOnDropped();
   Print("Script was dropped on the "+t);
```

另见

ChartXOnDropped, ChartYOnDropped

© 2000-2025, MetaQuotes Ltd.

# ChartXOnDropped

返回EA交易或者脚本下跌到的图表点x坐标。

```
int  ChartXOnDropped();
```

返回值

X 坐标值

注释

自左向右的X 轴方向。

示例：

```
   int X=ChartXOnDropped();
   int Y=ChartYOnDropped();
   Print("(X,Y) = ("+X+","+Y+")");
```

另见

ChartWindowOnDropped, ChartPriceOnDropped, ChartTimeOnDropped

© 2000-2025, MetaQuotes Ltd.

# ChartYOnDropped

返回EA交易或者脚本放置到的图表点Y坐标。

```
int  ChartYOnDropped();
```

返回值

Y坐标值。

注释

自上向下的Y轴方向。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得并打印使用鼠标把脚本放置到图表上的点的X和Y坐标
   int x=ChartXOnDropped();
   int y=ChartYOnDropped();
   PrintFormat("Script dropped to coordinates X = %d, Y = %d", x, y);
   /*
  结果：
   Script dropped to coordinates X = 429, Y = 114
   */
  }
```

另见

ChartWindowOnDropped, ChartPriceOnDropped, ChartTimeOnDropped

© 2000-2025, MetaQuotes Ltd.

# ChartSetSymbolPeriod

更改指定图表的交易品种和周期。函数不同步，例如，不等执行完成就发送命令。该命令添加到图表信息队列并且将在处理完之前所有命令后开始执行。

```
bool  ChartSetSymbolPeriod(
   long             chart_id,      // 图表 ID
   string           symbol,        // 交易品种名称
   ENUM_TIMEFRAMES  period         // 周期
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*交易品种*

[in] 图表交易品种。NULL 值意味着当前图表交易品种 (附上EA交易)。

*period*

[in] 图表周期 (时间表)。可以是ENUM_TIMEFRAMES值之一。0意味着当前图表周期。

返回值

如果该命令已添加到图表队列则返回true，否则false。若获得额外错误信息，需要调用函数GetLastError()。

注释

交易品种/周期的更改导致图表中EA交易的重新初始化。

通过相同的交易品种和时间周期调用ChartSetSymbolPeriod可以用来更新图表 (类似于客户端的刷新命令)。反过来，图表的更新还会重新计算所附的指标。因此，即使没有报价您仍然可以计算图表中的指标 (例如：周末)。

示例：

```
#define   SYMBOL    "GBPUSD"
#define   PERIOD    PERIOD_H1

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得当前图表ID, 交易品种和周期
   long             chart_id= ChartID();
   string           symbol  = Symbol();
   ENUM_TIMEFRAMES period  = Period();

//--- 在日志中报告使用设置在SYMBOL和PERIOD中的值替换当前图表的交易品种和周期
   PrintFormat("Change the %s symbol and the %s period of the chart %I64u to %s %s",
               symbol, TimeframeDescrioption(period), chart_id, SYMBOL, TimeframeDesc
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 改变图表交易品种和周期
   ChartSetSymbolPeriod(chart_id, SYMBOL, PERIOD);
   /*
   结果：
   Change the EURUSD symbol and the M1 period of the chart 133246248352168440 to GBPUS
   */
  }
//+------------------------------------------------------------------+
//| 返回图表周期的描述                                               |
//+------------------------------------------------------------------+
string TimeframeDescrioption(const ENUM_TIMEFRAMES timeframe)
  {
   return(StringSubstr(EnumToString(timeframe), 7));
  }
```

另见

图表交易品种，图表周期

© 2000-2025, MetaQuotes Ltd.

# ChartScreenShot

函数提供gif,png或bmp格式图表当前状态的屏幕截屏。

```
bool  ChartScreenShot(
   long             chart_id,                  // 图表 ID
   string           filename,                  // 交易品种名称
   int              width,                     // 宽度
   int              height,                    // 高度
   ENUM_ALIGN_MODE  align_mode=ALIGN_RIGHT     // 对齐类型
   );
```

参量

*chart_id*

[in] 图表 ID. 0 意味着当前图表。

*filename*

[in] 屏幕截图文件名。不能超过63个字符。屏幕截图文件位于directory\Files。

*width*

[in] 屏幕截图宽度像素。

*height*

[in] 屏幕截图高度像素。

*align_mode=ALIGN_RIGHT*

[in] 屏幕截图的有限输出模式。ENUM_ALIGN_MODE枚举的值。ALIGN_RIGHT 表示右边距对齐（从末端输出）。ALIGN_LEFT 表示左对齐。

返回值

若成功返回true，否则false。

注释

如果需要从图表中的某一点截图，首先用ChartNavigate()函数定位图表。如果截图水平大小小于图表窗口，根据align_mode设置，会选择图表窗口的右边或者左边输出。

例如：

```
#property description "The Expert Advisor demonstrates how to create a series of scree
#property description "chart using the ChartScreenShot() function. For convenience, th
#property description "shown on the chart. The height and width of images is defined t

#define        WIDTH  800      // 图像宽度，调用ChartScreenShot()
#define        HEIGHT 600      // 图像高度，调用ChartScreenShot()

//--- 输入参数
input int      pictures=5;     // 序列中的图像数量
int            mode=-1;        // -1 表示移到图表右边界，1 - 移到左边
int            bars_shift=300;// 使用ChartNavigate()滚动图表时的柱形数量
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 禁止图表自动滚动
   ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- 设置图表右边界转移
   ChartSetInteger(0,CHART_SHIFT,true);
//--- 显示蜡烛图图表
   ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
   Print("Preparation of the Expert Advisor is completed");
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 显示函数名称，调用时间和事件标识符
   Print(__FUNCTION__,TimeCurrent(),"   id=",id,"   mode=",mode);
//--- 处理 CHARTEVENT_CLICK 事件 ("鼠标点击图表")
   if(id==CHARTEVENT_CLICK)
     {
      //--- 从图表边界初始移动
      int pos=0;
      //--- 左侧图表边界操作
      if(mode>0)
        {
         //--- 滚动图表至左侧边界
         ChartNavigate(0,CHART_BEGIN,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- 准备在图表上显示的文本和文件名
            string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif";
            //--- 在图表上显示名称作为注释
            Comment(name);
            //--- 在terminal_directory\MQL5\Files\文件中保存图表截图
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
```

© 2000-2025, MetaQuotes Ltd.

```
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- 给予用户时间查看新的图表部分
            Sleep(3000);
            //--- 从当前位置bars_shift柱滚动图表到右侧
            ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
           }
         //--- 改变模式到相反方向
         mode*=-1;
        }
      else // 图表右侧边界的操作
        {
         //--- 滚动图表至右侧边界
         ChartNavigate(0,CHART_END,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- 准备在图表上显示的文本和文件名
            string name="ChartScreenShot"+"CHART_END"+string(pos)+".gif";
            //--- 在图表上显示名称作为注释
            Comment(name);
            //--- 在terminal_directory\MQL5\Files\文件中保存图表截图
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_RIGHT))
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- 给予用户时间查看新的图表部分
            Sleep(3000);
            //--- 从当前位置bars_shift柱滚动图表到右侧
            ChartNavigate(0,CHART_CURRENT_POS,-bars_shift);
           }
         //--- 改变模式到相反方向
         mode*=-1;
        }
     } // CHARTEVENT_CLICK 事件处理结束
//--- OnChartEvent()句柄末端
  }
```

另见

ChartNavigate(), 资源

© 2000-2025, MetaQuotes Ltd.

# 交易函数

该组函数用于管理交易活动。

在您开始学习平台提供的交易功能之前，您需要清楚地理解基本条款：订单，成交和仓位：

- 一笔订单就是一条指令，令经纪商买入或卖出一种金融工具。有两种主要的订单类型：市价和待定。此外，还有特别的止盈和止损价位。
- 一笔成交是一种金融证券的商业互换（买入或卖出）。在采购价（Ask）执行买入，在供给价（Bid）执行卖出。成交可以是执行市价单，或触发挂单的结果。注意在一些情况下，执行一笔订单可以有若干笔成交。
- 一笔仓位是一份交易合同，即买入或卖出一种金融工具合约的数量。一笔多头仓位是买入的金融证券，期望证券价格升高。一笔空头仓位是一份证券供应合同，期望价格在未来下跌。

关于交易操作的一般信息可在客户端帮助中找到。

交易函数在EA交易或者脚本中使用，交易函数只有在脚本或EA交易的"允许在线交易"多选框允许时调用。

根据 交易权限 版块描述的不同因素允许或禁止交易。

| 函数 | 功能 |
|---|---|
| OrderCalcMargin | 用入金货币，计算指定订单类型所需的保证金 |
| OrderCalcProfit | 以入金货币计算基于传递参量的利润 |
| OrderCheck | 检测是否有足够资金执行所需 交易操作. |
| OrderSend | 发送 交易需求 到服务器 |
| OrderSendAsync | 不同步发送 交易请求 无需等待交易服务器的交易回应 |
| PositionsTotal | 返回持仓数量 |
| PositionGetSymbol | 返回与持仓一致的交易品种 |
| PositionSelect | 选择一个持仓进行下一步工作 |
| PositionSelectByTicket | 通过指定的单号选择工作持仓 |
| PositionGetDouble | 返回持仓要求属性（双精度） |
| PositionGetInteger | 返回持仓要求属性（日期时间或者整型） |
| PositionGetString | 返回持仓要求属性（字符串） |
| PositionGetTicket | 返回持仓列表中指定索引的持仓单号 |
| OrdersTotal | 返回订单数 |
| OrderGetTicket | 返回相应订单的订单号 |
| OrderSelect | 为下一步工作选择一订单 |
| OrderGetDouble | 返回订单要求属性（双精度型） |
| OrderGetInteger | 返回订单要求属性（日期时间或者整型） |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| OrderGetString | 返回订单要求属性（字符串） |
| HistorySelect | 为服务器时间的指定周期检索交易和订单历史记录 |
| HistorySelectByPosition | 请求指定仓位标识符交易的历史记录 |
| HistoryOrderSelect | 为下一步工作选择历史订单 |
| HistoryOrdersTotal | 返回历史记录中订单数 |
| HistoryOrderGetTicket | 返回历史记录中相关订单的订单号 |
| HistoryOrderGetDouble | 返回（双精度）历史记录中订单需求属性 |
| HistoryOrderGetInteger | 返回（日期时间或者整型）历史记录中订单需求属性 |
| HistoryOrderGetString | 返回（字符串）历史记录中订单需求属性 |
| HistoryDealSelect | 历史记录中选择一个交易便于通过适当函数进一步调用它。 |
| HistoryDealsTotal | 返回历史记录交易数 |
| HistoryDealGetTicket | 返回历史相关交易订单号 |
| HistoryDealGetDouble | 返回历史记录中交易需求属性（双精度） |
| HistoryDealGetInteger | 返回历史记录中交易需求属性（时间日期和整型） |
| HistoryDealGetString | 返回历史记录中交易需求属性（字符串） |

© 2000-2025, MetaQuotes Ltd.

# OrderCalcMargin

函数计算指定订单类型的需要保证金，在活期账户，当前的市场环境不能计算到当前的挂单和持仓。允许计算的保证金需要交易操作计划，值是返回的当前账户的值。

```
bool  OrderCalcMargin(
   ENUM_ORDER_TYPE       action,           // 订单类型
   string                symbol,           // 交易品种名称
   double                volume,           // 交易量
   double                price,            // 开盘价
   double&               margin            // 为获取保证金值的值
   );
```

参量

*action*

[in] 订单类型，可以是 ENUM_ORDER_TYPE 值中的一个。

*symbol*

[in] 交易品种名称。

*volume*

[in] 交易操作成交量。

*price*

[in] 开盘价。

*margin*

[out]要求保证金的值的变量成功执行到相应函数中去，计算会执行活期账户的开盘价而非交易订单，保证金值有很多元素，在不同的交易环境中相互区别。

返回值

如果成功，函数返回true，否则返回false，为了获得有关的 错误 信息，调用 GetLastError() 函数。

示例：

```
#define   VOLUME     1.0   // 订单交易量
#define   DEVIATION  100   // 用于设置挂单的距离
#define   STOP_LIMIT 50    // 订单止损限价的距离

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   int array_types[8]={ORDER_TYPE_BUY,
                       ORDER_TYPE_SELL,
                       ORDER_TYPE_BUY_LIMIT,
                       ORDER_TYPE_SELL_LIMIT,
```

© 2000-2025, MetaQuotes Ltd.

```
                        ORDER_TYPE_BUY_STOP,
                        ORDER_TYPE_SELL_STOP,
                        ORDER_TYPE_BUY_STOP_LIMIT,
                        ORDER_TYPE_SELL_STOP_LIMIT };

//--- 根据订单类型数组进行循环
   for(int i=0; i<(int)array_types.Size(); i++)
     {
      //--- 根据订单类型，取得 ORDER_TYPE_BUY 或 ORDER_TYPE_SELL 类型
      ENUM_ORDER_TYPE type=MarketOrderByOrderType((ENUM_ORDER_TYPE)array_types[i]);

      //--- 根据订单类型，取得价格
      double price=PriceByOrderType(_Symbol, type);

      //--- 根据 'action' 中指定的订单类型取得所需的保证金金额
      double margin=EMPTY_VALUE;
      ResetLastError();
      if(!OrderCalcMargin(type, _Symbol, VOLUME, price, margin))
        {
         Print("OrderCalcMargin() failed. Error ",GetLastError());
         continue;
        }
      //--- 在日志中打印结果
      PrintFormat("Margin required for %.2f %s order at price %.*f on %s symbol: %.2f
     }
   /*
   结果：
   Margin required for 1.00 Buy order at price 1.31668 on GBPUSD symbol: 1316.68 USD
   Margin required for 1.00 Sell order at price 1.31661 on GBPUSD symbol: 1316.61 USD
   Margin required for 1.00 Buy Limit order at price 1.31568 on GBPUSD symbol: 1315.68
   Margin required for 1.00 Sell Limit order at price 1.31761 on GBPUSD symbol: 1317.6
   Margin required for 1.00 Buy Stop order at price 1.31768 on GBPUSD symbol: 1317.68
   Margin required for 1.00 Sell Stop order at price 1.31561 on GBPUSD symbol: 1315.61
   Margin required for 1.00 Buy Stop Limit order at price 1.31718 on GBPUSD symbol: 13
   Margin required for 1.00 Sell Stop Limit order at price 1.31611 on GBPUSD symbol: 1
   */
  }
//+------------------------------------------------------------------+
//| 返回买入或卖出订单类型                                           |
//+------------------------------------------------------------------+
ENUM_ORDER_TYPE MarketOrderByOrderType(ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY  : case ORDER_TYPE_BUY_LIMIT  : case ORDER_TYPE_BUY_STOP  :
        return(ORDER_TYPE_BUY);
      case ORDER_TYPE_SELL : case ORDER_TYPE_SELL_LIMIT : case ORDER_TYPE_SELL_STOP :
        return(ORDER_TYPE_SELL);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
    return(WRONG_VALUE);
   }
//+------------------------------------------------------------------+
//| 根据订单类型返回开单价格                                         |
//+------------------------------------------------------------------+
double PriceByOrderType(const string symbol, const ENUM_ORDER_TYPE order_type)
   {
    int     digits=0;
    double  point=0;
    MqlTick tick={};

//--- 取得交易品种每个点(Point)的值
    ResetLastError();
    if(!SymbolInfoDouble(symbol, SYMBOL_POINT, point))
      {
       Print("SymbolInfoDouble() failed. Error ", GetLastError());
       return 0;
      }

//--- 取得交易品种小数位(Digits)的值
    long value=0;
    if(!SymbolInfoInteger(symbol, SYMBOL_DIGITS, value))
      {
       Print("SymbolInfoInteger() failed. Error ", GetLastError());
       return 0;
      }
    digits=(int)value;

//--- 取得交易品种的最新价格
    if(!SymbolInfoTick(symbol, tick))
      {
       Print("SymbolInfoTick() failed. Error ", GetLastError());
       return 0;
      }

//--- 返回根据订单类型的价格
    switch(order_type)
      {
       case ORDER_TYPE_BUY              :  return(tick.ask);
       case ORDER_TYPE_SELL             :  return(tick.bid);
       case ORDER_TYPE_BUY_LIMIT        :  return(NormalizeDouble(tick.ask - DEVIATION
       case ORDER_TYPE_SELL_LIMIT       :  return(NormalizeDouble(tick.bid + DEVIATION
       case ORDER_TYPE_BUY_STOP         :  return(NormalizeDouble(tick.ask + DEVIATION
       case ORDER_TYPE_SELL_STOP        :  return(NormalizeDouble(tick.bid - DEVIATION
       case ORDER_TYPE_BUY_STOP_LIMIT   :  return(NormalizeDouble(tick.ask + DEVIATION
       case ORDER_TYPE_SELL_STOP_LIMIT  :  return(NormalizeDouble(tick.bid - DEVIATION
       default                          :  return(0);
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE order_type)
  {
   switch(order_type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      case ORDER_TYPE_CLOSE_BY         :  return("Close By");
      default                          :  return("Unknown order type");
     }
  }
//+------------------------------------------------------------------+
```

另见

OrderSend(), 订单属性, 交易操作类型

© 2000-2025, MetaQuotes Ltd.

# OrderCalcProfit

函数为当前账户计算利润，在当前的交易市场条件下，以参量为基础传递。函数使用交易操作的结构进行再评估，值是返回的当前账户的值。

```
bool  OrderCalcProfit(
   ENUM_ORDER_TYPE       action,          // 订单类型 (ORDER_TYPE_BUY 或者 ORDER_TYPE_S
   string                symbol,          // 交易品种名称
   double                volume,          // 交易量
   double                price_open,      // 开盘价
   double                price_close,     // 收盘价
   double&               profit           // 为获得利润值的变量
   );
```

参量

*action*

[in] 订单类型，可以是 ENUM_ORDER_TYPE 项目里ORDER_TYPE_BUY 或 ORDER_TYPE_SELL值中的一个。

*symbol*

[in] 交易品种名称。

*volume*

[in] 交易操作成交量。

*price_open*

[in] 开盘价。

*price_close*

[in] 收盘价。

*profit*

[out] 计算利润的变量值在函数成功执行后编写进去，估计利润值依据许多因素，并在不同的市场环境中加以区别。

返回值

如果成功，函数返回true,否则返回false。如果指定无效订单类型，函数返回false，为了获得错误信息，可以调用 GetLastError()。

示例：

```
#define   VOLUME      1.0   // 订单交易量
#define   DEVIATION   100   // 平仓价格之前的点数

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   string currency_profit=SymbolInfoString(_Symbol,SYMBOL_CURRENCY_PROFIT);    // 利润货
   double close_price=ChartPriceOnDropped(); // 价格坐标，对应着使用鼠标拖曳脚本的点位，用于

//---
   for(int i=0; i<=ORDER_TYPE_SELL; i++)
     {
      ENUM_ORDER_TYPE order_type=(ENUM_ORDER_TYPE)i;                  // 订单类型：0 - 买
      double open_price=PriceOpenByOrderType(_Symbol, order_type);    // 下单价格：对于买

      //--- 如果未获得开单价格，继续循环
      if(open_price==0)
         continue;

      //--- 如果平仓价格为0（脚本不是通过拖曳到图表上而运行的），计算价格
      if(close_price==0)
         close_price=(order_type==ORDER_TYPE_BUY ? open_price + DEVIATION * _Point : c

      //--- 根据传入的参数计算当前账户和市场环境下的利润大小
      double profit=0;
      ResetLastError();
      if(!OrderCalcProfit(order_type, _Symbol, VOLUME, open_price, close_price, profit
        {
         Print("OrderCalcProfit() failed. Error ", GetLastError());
         continue;
        }

      //--- 在日志中打印收到的利润值
      PrintFormat("Estimated profit for %.2f %s position on %s with opening price of %
                  VOLUME, OrderTypeDescription(order_type), _Symbol, _Digits, open_pri

     }
   /*
   结果：
   Estimated profit for 1.00 Buy position on EURUSD with opening price of 1.10757 and
   Estimated profit for 1.00 Sell position on EURUSD with opening price of 1.10753 and
   */
  }
//+------------------------------------------------------------------+
//| 根据订单类型返回开单价格                                         |
//+------------------------------------------------------------------+
double PriceOpenByOrderType(const string symbol, const ENUM_ORDER_TYPE order_type)
  {
   MqlTick tick={};

//--- 取得交易品种的最新价格
   if(!SymbolInfoTick(symbol, tick))
     {
      Print("SymbolInfoTick() failed. Error ", GetLastError());
      return 0;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 返回根据订单类型的价格
   switch(order_type)
      {
       case ORDER_TYPE_BUY              :  return(tick.ask);
       case ORDER_TYPE_SELL             :  return(tick.bid);
       default                          :  return(0);
      }
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE order_type)
  {
   switch(order_type)
      {
       case ORDER_TYPE_BUY              :  return("Buy");
       case ORDER_TYPE_SELL             :  return("Sell");
       default                          :  return("Unknown order type");
      }
  }
```

另见

OrderSend(), 订单属性, 交易操作类型

© 2000-2025, MetaQuotes Ltd.

# OrderCheck

OrderCheck()函数检测是否有足够的钱执行需要的交易操作。检测结果在MqlTradeCheckResult结构域中。

```
bool  OrderCheck(
   MqlTradeRequest&        request,      // 请求结构
   MqlTradeCheckResult&    result        // 结果结构
   );
```

参量

*request*

[in] 执行 MqlTradeRequest 结构类型，该类型表述必要的操作活动。

*result*

[in,out] 指针指向 MqlTradeCheckResult 结构类型，显示检测结果。

返回值

如果操作时资金不足，或者错误填满了参量，函数返回false，如果成功检测了结构基础（检测指针），返回true-**这并不表示要求的交易操作已经成功执行了**。更多函数执行结果的细节描述，都分析在*result*结构域中。

为了获得有关错误的信息，调用 GetLastError() 函数。

示例：

```
#define   DEVIATION      5              // 允许的价格偏差
#define   VOLUME         1.0            // 订单交易量
#define   EXPERT_MAGIC   123            // 幻数
#define   DIRECTION      ORDER_TYPE_BUY // 开启的仓位方向 (ORDER_TYPE_BUY 或 ORDER_TYPE_S

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明并初始化请求,验证和结果结构
   MqlTradeRequest      request={};
   MqlTradeCheckResult check  ={};
   MqlTradeResult       result ={};

//--- 准备交易请求参数
   PrepareRequest(_Symbol, DIRECTION, VOLUME, request);

//--- 检查交易请求的参数
   ResetLastError();
   bool res=OrderCheck(request, check);
   if(!res)
     {
      PrintFormat("Trade request verification completed with error %d\nServer retcode
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }

//--- 交易请求检查成功 - 显示交易请求验证结构栏位的描述
   Print("Trade request verification completed successfully");
   MqlTradeCheckResultPrint(check, 14);

//--- 发送交易请求
   if(!OrderSend(request, result))
      Print("OrderSend error ", GetLastError());     // 如果无法发送请求, 显示错误代码

//--- 有关操作的信息
   PrintFormat("Trade request result: retcode=%u, deal=%I64u, order=%I64u", result.ret
   /*
   result with disabled auto trading in the client terminal:
   Trade request verification completed with error 4752
   Server retcode: 10027, comment: AutoTrading disabled by client

   enable auto trading and check again on a closed market:
   Experts   automated trading is enabled
   Trade request verification completed successfully
   Retcode:      0
   Balance:      10779.50 USD
   Equity:       10779.50 USD
   Profit:       0.00 USD
   Margin:       1104.79 USD
   Margin free:  9674.71 USD
   Margin level: 975.71 %
   Comment:      Done
   OrderSend error 4756
   Trade request result: retcode=10018, deal=0, order=0

   check on the open market:
   Trade request verification completed successfully
   Retcode:      0
   Balance:      10779.50 USD
   Equity:       10779.50 USD
   Profit:       0.00 USD
   Margin:       110.46 USD
   Margin free:  10669.04 USD
   Margin level: 9758.74 %
   Comment:      Done
   Trade request result: retcode=10009, deal=2777010968, order=2802818813
   */
  }
//+------------------------------------------------------------------+
//| 准备用于交易请求的参数                                           |
//+------------------------------------------------------------------+
void PrepareRequest(const string symbol, const ENUM_ORDER_TYPE order_type, const doubl
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   ENUM_ORDER_TYPE type=(DIRECTION !=ORDER_TYPE_BUY ? ORDER_TYPE_SELL : DIRECTION);
   double price=(DIRECTION==ORDER_TYPE_BUY ? SymbolInfoDouble(Symbol(), SYMBOL_ASK) :
//--- 请求参数
   request.action    = TRADE_ACTION_DEAL; // 交易操作类型
   request.symbol    = symbol;            // 交易品种
   request.volume    = volume;            // 交易量
   request.type      = type;              // 订单类型
   request.price     = price;             // 开单价格
   request.deviation = DEVIATION;         // 允许的价格偏差
   request.magic     = EXPERT_MAGIC;      // 订单幻数
  }
//+------------------------------------------------------------------+
//| 在日志中打印交易请求                                             |
//| 验证结果的栏位                                                   |
//+------------------------------------------------------------------+
void MqlTradeCheckResultPrint(const MqlTradeCheckResult &check, const uint header_widt
  {
//--- 取得账户货币和账户货币的小数点位数
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   int    digits  =(int)AccountInfoInteger(ACCOUNT_CURRENCY_DIGITS);

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0，则宽度为标题文字行的大小+1
   string header="Retcode:";
   uint w=(header_width==0 ? header.Length()+1 : header_width);
//--- 在日志中以指定的标题宽度显示返回代码
   PrintFormat("%-*s%-u", w, header, check.retcode);

//--- 在日志中显示执行交易操作后的余额值
   header="Balance:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.balance, currency);

//--- 在日志中显示执行交易后的净值数值
   header="Equity:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.equity, currency);

//--- 在日志中显示执行了交易操作后的浮动利润值
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.profit, currency);

//--- 在日志中显示必要交易操作所需的保证金金额
   header="Margin:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.margin, currency);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在日志中显示执行了一项交易操作后剩余的净值数值
   header="Margin free:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.margin_free, currency);

//--- 在日志中显示在完成所请求的交易操作后的保证金水平
   header="Margin level:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.2f %%", w, header, check.margin_level);

//--- 在日志中显示相应代码的注释和错误描述
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-s", w, header, check.comment);
  }
```

另见

OrderSend(), 交易操作类型, 交易请求结构, 检测结果请求结构, 交易请求结果结构

© 2000-2025, MetaQuotes Ltd.

# OrderSend

通过向服务器发送请求，OrderSend()函数用来执行交易操作操作。

```
bool  OrderSend(
   MqlTradeRequest&  request,      // query structure
   MqlTradeResult&   result        // structure of the answer
   );
```

参量

*request*

[in] 指针指向 MqlTradeRequest 结构类型操作，描述成客户的交易活动。

*result*

[in,out]指针指向MqlTradeResult结构类型操作，描述成功完成后的交易操作结构（如果真值可以返回）。

返回值

成功检测基础结构（索引检测）返回真值-这并不是交易操作成功执行的标志。更多函数执行结果的细节描述，都分析在*result*结构域中。

注释

交易需要通过多个步骤检测交易服务器，第一，检测是否所有要求操作*request*参量都正确填满，如果没有错误，服务器接收下一步订单，如果命令通过服务器成功接收，OrderSend()函数返回真值。

在向服务器发送请求之前推荐检验请求，为了检验请求，使用OrderCheck()函数。检验是否有足够的资金来执行交易操作，并在交易请求检测结果中返回许多有用的参量：

- 返回码 在检测请求中包括错误信息；
- 在交易操作执行后显示账户余额；
- 在交易操作执行后显示产权值；
- 在交易操作执行后显示浮点类型值；
- 交易操作所需保证金；
- 在交易操作执行后可用产权数量；
- 在交易操作执行后建立的保证金水平；
- 注释回应代码，错误描述。

When sending a market order (MqlTradeRequest.action=TRADE_ACTION_DEAL), the successful result of the OrderSend() function does not mean that the order has been executed (appropriate trades have been performed). In this case, 'true' means only that the order has been successfully placed in the trading system for further execution. The trade server can fill in the *deal or order* field values *in the returned* result tructure, if it is aware of these data when forming a response to an OrderSend() call. Generally, event(s) of executing trades corresponding to an order may happen after sending a response to the OrderSend() call. Therefore, for any type of a trade request, when receiving the OrderSend() execution result, we should first check the *retcode* trade server response code and the *retcode_external* external system response code (if necessary) available in the obtained result tructure.

每个接收到的订单存储在交易服务器中排队处理，直到可以执行的条件发生：

- 临界期，

© 2000-2025, MetaQuotes Ltd.

- 反面要求的外观，
- 当执行价格出现时命令执行，
- 收到的取消订单的命令。

订单执行时，交易服务器向客户端发送关于交易事件程序发生的信息，可以通过OnTrade()函数执行。

在通过OrderSend()函数发送的服务器上执行交易请求的结果可以通过OnTradeTransaction 处理程序跟踪。应该注意，当执行交易请求时，将会多次调用OnTradeTransaction处理程序。

例如，当发送市场买入订单时，它会被处理，为账户创建一个适当的买入订单，然后执行订单并从持仓订单列表中移除，然后它被添加到订单历史记录，相应的交易也被添加到历史记录并创建新持仓。OnTradeTransaction函数将会为这里每个事件进行调用。

示例：

```
//--- ORDER_MAGIC值
input long order_magic=55555;
//+------------------------------------------------------------------+
//| 启动函数的脚本程序                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 确保账户是样本
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("Script operation is not allowed on a live account!");
      return;
     }
//--- 下订单或者删除订单
   if(GetOrdersTotalByMagic(order_magic)==0)
     {
      //--- 无当前订单 - 下订单
      uint res=SendRandomPendingOrder(order_magic);
      Print("Return code of the trade server ",res);
     }
   else // 有订单 - 删除订单
     {
      DeleteAllOrdersByMagic(order_magic);
     }
//---
  }
//+------------------------------------------------------------------+
//| 接收当前带有指定的ORDER_MAGIC的订单号                              |
//+------------------------------------------------------------------+
int GetOrdersTotalByMagic(long const magic_number)
  {
   ulong order_ticket;
   int total=0;
//--- 检查通过所有挂单
   for(int i=0;i<OrdersTotal();i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      if((order_ticket=OrderGetTicket(i))>0)
         if(magic_number==OrderGetInteger(ORDER_MAGIC)) total++;
//---
   return(total);
  }
//+------------------------------------------------------------------+
//| 删除所有带有指定ORDER_MAGIC的挂单                                |
//+------------------------------------------------------------------+
void DeleteAllOrdersByMagic(long const magic_number)
  {
   ulong order_ticket;
//--- 检查所有挂单
   for(int i=OrdersTotal()-1;i>=0;i--)
      if((order_ticket=OrderGetTicket(i))>0)
         //--- 带有恰当ORDER_MAGIC的订单
         if(magic_number==OrderGetInteger(ORDER_MAGIC))
           {
            MqlTradeResult result={};
            MqlTradeRequest request={};
            request.order=order_ticket;
            request.action=TRADE_ACTION_REMOVE;
            OrderSend(request,result);
            //--- 编写服务器回复到日志
            Print(__FUNCTION__,": ",result.comment," reply code ",result.retcode);
           }
//---
  }
//+------------------------------------------------------------------+
//| 随机设置挂单                                                     |
//+------------------------------------------------------------------+
uint SendRandomPendingOrder(long const magic_number)
  {
//--- 准备请求
   MqlTradeRequest request={};
   request.action=TRADE_ACTION_PENDING;          // 设置挂单
   request.magic=magic_number;                   // ORDER_MAGIC
   request.symbol=_Symbol;                       // 交易品种
   request.volume=0.1;                           // 0.1为单位的交易量
   request.sl=0;                                 // 没有指定止损价位
   request.tp=0;                                 // 没有指定盈利价位
//--- 形成订单类型
   request.type=GetRandomType();                 // 订单类型
//--- 形成挂单价格
   request.price=GetRandomPrice(request.type);   // 开盘价
//--- 发送交易请求
   MqlTradeResult result={};
   OrderSend(request,result);
//--- 编写服务器回复到日志
   Print(__FUNCTION__,":",result.comment);
```

© 2000-2025, MetaQuotes Ltd.

```
   if(result.retcode==10016) Print(result.bid,result.ask,result.price);
//--- 返回交易服务器回复的代码
   return result.retcode;
  }
//+------------------------------------------------------------------+
//| 返回随机挂单类型                                                 |
//+------------------------------------------------------------------+
ENUM_ORDER_TYPE GetRandomType()
  {
   int t=MathRand()%4;
//---   0<=t<4
   switch(t)
     {
      case(0):return(ORDER_TYPE_BUY_LIMIT);
      case(1):return(ORDER_TYPE_SELL_LIMIT);
      case(2):return(ORDER_TYPE_BUY_STOP);
      case(3):return(ORDER_TYPE_SELL_STOP);
     }
//--- 不正确值
   return(WRONG_VALUE);
  }
//+------------------------------------------------------------------+
//| 返回随机价格                                                     |
//+------------------------------------------------------------------+
double GetRandomPrice(ENUM_ORDER_TYPE type)
  {
   int t=(int)type;
//--- 交易品种止损水平
   int distance=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
//--- 接收上一个订单号数据
   MqlTick last_tick={};
   SymbolInfoTick(_Symbol,last_tick);
//--- 按照类型计算价格
   double price;
   if(t==2 || t==5) // ORDER_TYPE_BUY_LIMIT or ORDER_TYPE_SELL_STOP
     {
      price=last_tick.bid; // 不同于卖价
      price=price-(distance+(MathRand()%10)*5)*_Point;
     }
   else             // ORDER_TYPE_SELL_LIMIT or ORDER_TYPE_BUY_STOP
     {
      price=last_tick.ask; // 不同于买价
      price=price+(distance+(MathRand()%10)*5)*_Point;
     }
//---
   return(price);
  }
```

另 见

© 2000-2025, MetaQuotes Ltd.

交易操作类型, 交易请求结构, 请求检测结果结构, 交易请求结果结构

© 2000-2025, MetaQuotes Ltd.

# OrderSendAsync

OrderSendAsync() 函数用于指导不同步的 交易操作 ，无需等待交易服务器对发送请求的回应。该函数专为高频交易设计，根据交易算法，无法接受浪费时间等候来自服务器的回应。

```
bool  OrderSendAsync(
   MqlTradeRequest&  request,      // 请求架构
   MqlTradeResult&   result        // 请求架构
   );
```

参数

*request*

[in] 描述客户交易行为的MqlTradeRequest 类型架构的指针。

*result*

[in,out] 函数成功执行时（如果返回true），描述交易操作结果的MqlTradeResult类型结构的指针。

返回值

如果请求发送到交易服务器返回true。如果请求没有发送，它返回false。如果请求被发送，*result* 变量中反馈代码包含TRADE_RETCODE_PLACED 值（代码 10008）-"已下单"。成功执行意味着只是发送的事实，但是不保证请求到达交易服务器和已被接受处理。当处理接收的请求时，交易服务器向客户端发送回应，通知引起生成交易事件的持仓当前状态，订单和交易的变化。

在通过OrderSendAsync()函数发送的服务器上执行交易请求的结果可以通过OnTradeTransaction处理程序跟踪。应该注意，当执行交易请求时，将会多次调用OnTradeTransaction处理程序。

例如，当发送市场买入订单时，它会被处理，为账户创建一个适当的买入订单，然后执行订单并从持仓订单列表中移除，然后它被添加到订单历史记录，相应的交易也被添加到历史记录并创建新持仓。OnTradeTransaction函数将会为这里每个事件进行调用。若要获得这种数据，应该分析函数参数：

- **trans** - 该参数获取描述应用到交易账户的交易事务的 MqlTradeTransaction 架构；
- **request** - 该参数获取描述一个交易事务得出的交易请求的 MqlTradeRequest 架构；
- **result** - 该参数获取描述交易请求执行结果的 MqlTradeResult 架构。

注意

根据意图和参数，该函数类似OrderSend()，但是又不一样，它是非同步的，例如，等候函数执行结果时不保持程序操作。您可以使用EA交易样本比较这两个函数交易操作的比率。

例如：

© 2000-2025, MetaQuotes Ltd.

```
#property description "Expert Advisor for sending trade requests "
                      " using OrderSendAsync() function.\r\n"
#property description "Handling trading events using"
                      " OnTrade() and OnTradeTransaction() handler functions is displa
#property description "Expert Advisor parameters allow setting Magic Number"
                      " (unique ID) "
#property description "and the mode of displaying messages in Experts log. All details
//--- 输入参数
input int  MagicNumber=1234567;      // EA交易 ID
input bool DescriptionModeFull=true; // 详细的输出模式
//--- HistorySelect()调用使用的变量
datetime history_start;
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 检查是否允许自动交易
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
     {
      Alert("Autotrading in the terminal is disabled, Expert Advisor will be removed."
      ExpertRemove();
      return(-1);
     }
//--- 不能交易真实账户
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("Expert Advisor cannot trade on a real account!");
      ExpertRemove();
      return(-2);
     }
//--- 检查是否可以在该账户交易 例如，使用只读密码时不能交易）
   if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
     {
      Alert("Trading on this account is disabled");
      ExpertRemove();
      return(-3);
     }
//--- 节省启动EA交易的时间来接收交易历史记录
   history_start=TimeCurrent();
//---
   CreateBuySellButtons();
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 删除所有图形对象
   ObjectDelete(0,"Buy");
   ObjectDelete(0,"Sell");
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction 函数                                            |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 标题命名在交易事件处理程序函数之后
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- 接收事务类型作为枚举值
   ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
//--- 如果事务是请求处理的结果
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      //--- 显示事务名称
      Print(EnumToString(type));
      //--- 然后显示已处理请求的字符串描述
      Print("------------RequestDescription\r\n",
            RequestDescription(request,DescriptionModeFull));
      //--- 并显示请求结果的描述
      Print("------------ ResultDescription\r\n",
            TradeResultDescription(result,DescriptionModeFull));
     }
   else // 显示另一类型事务的完整描述
     {
      Print("------------ TransactionDescription\r\n",
            TransactionDescription(trans,DescriptionModeFull));
     }
//---
  }
//+------------------------------------------------------------------+
//| 交易函数                                                         |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- 存储交易账户状态的静态成员
   static int prev_positions=0,prev_orders=0,prev_deals=0,prev_history_orders=0;
//--- 请求交易历史记录
   bool update=HistorySelect(history_start,TimeCurrent());
   PrintFormat("HistorySelect(%s , %s) = %s",
               TimeToString(history_start),TimeToString(TimeCurrent()),(string)update);
//--- 标题命名在交易事件处理程序函数之后
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- 显示处理程序名称和处理时期订单数量
   int curr_positions=PositionsTotal();
   int curr_orders=OrdersTotal();
   int curr_deals=HistoryOrdersTotal();
   int curr_history_orders=HistoryDealsTotal();
//--- 用括号显示订单数量，持仓，成交，以及改变
   PrintFormat("PositionsTotal() = %d (%+d)",
               curr_positions,(curr_positions-prev_positions));
   PrintFormat("OrdersTotal() = %d (%+d)",
               curr_orders,curr_orders-prev_orders);
   PrintFormat("HistoryOrdersTotal() = %d (%+d)",
               curr_deals,curr_deals-prev_deals);
   PrintFormat("HistoryDealsTotal() = %d (%+d)",
               curr_history_orders,curr_history_orders-prev_history_orders);
//--- 插入字符串断点，更便于查看日志
   Print("");
//--- 保存账户状态
   prev_positions=curr_positions;
   prev_orders=curr_orders;
   prev_deals=curr_deals;
   prev_history_orders=curr_history_orders;
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent 函数                                                  |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 处理 CHARTEVENT_CLICK 事件 ("点击图表")
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("=> ",__FUNCTION__,": sparam = ",sparam);
      //--- 成交的最小交易量
      double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
      //--- 如果按下"买入"按钮，则买入
      if(sparam=="Buy")
        {
         PrintFormat("Buy %s %G lot",_Symbol,volume_min);
         BuyAsync(volume_min);
         //--- 不按下该按钮
         ObjectSetInteger(0,"Buy",OBJPROP_STATE,false);
        }
      //--- 如果按下"卖出"按钮，则卖出
      if(sparam=="Sell")
        {
         PrintFormat("Sell %s %G lot",_Symbol,volume_min);
         SellAsync(volume_min);
         //--- 不按下该按钮
         ObjectSetInteger(0,"Sell",OBJPROP_STATE,false);
        }
      ChartRedraw();
     }
//---
  }
//+------------------------------------------------------------------+
//| 返回交易事务的文本描述                                           |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans,
                              const bool detailed=true)
  {
//--- 准备从函数返回的字符串
   string desc=EnumToString(trans.type)+"\r\n";
//--- 在详细模式添加所有可能的数据
   if(detailed)
     {
      desc+="Symbol: "+trans.symbol+"\r\n";
      desc+="Deal ticket: "+(string)trans.deal+"\r\n";
      desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
      desc+="Order ticket: "+(string)trans.order+"\r\n";
      desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
      desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
      desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
      desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
      desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
      desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
     }
//--- 返回接收的字符串
   return desc;
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 返回交易请求的文本描述                                                   |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request,
                          const bool detailed=true)
  {
//--- 准备从函数返回的字符串
   string desc=EnumToString(request.action)+"\r\n";
//--- 在详细模式添加所有可用的数据
   if(detailed)
     {
      desc+="Symbol: "+request.symbol+"\r\n";
      desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
      desc+="Order ticket: "+(string)request.order+"\r\n";
      desc+="Order type: "+EnumToString(request.type)+"\r\n";
      desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
      desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
      desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
      desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
      desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
      desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
      desc+="Comment: "+request.comment+"\r\n";
     }
//--- 返回接收的字符串
   return desc;
  }
//+------------------------------------------------------------------+
//| 返回请求处理结果的文本描述                                               |
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result,
                              const bool detailed=true)
  {
//--- 准备从函数返回的字符串
   string desc="Retcode "+(string)result.retcode+"\r\n";
//--- 在详细模式添加所有可用的数据
   if(detailed)
     {
      desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
      desc+="Order ticket: "+(string)result.order+"\r\n";
      desc+="Deal ticket: "+(string)result.deal+"\r\n";
      desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
      desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
      desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
      desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
      desc+="Comment: "+result.comment+"\r\n";
     }
//--- 返回接收的字符串
   return desc;
  }
//+------------------------------------------------------------------+
//| 为买入和卖出创建两个按钮                                                 |
//+------------------------------------------------------------------+
void CreateBuySellButtons()
  {
//--- 检查名称为 "买入"的对象
   if(ObjectFind(0,"Buy")>=0)
     {
      //--- 如果发现的对象不是按钮，请删除
      if(ObjectGetInteger(0,"Buy",OBJPROP_TYPE)!=OBJ_BUTTON)
```

© 2000-2025, MetaQuotes Ltd.

```
         ObjectDelete(0,"Buy");
      }
   else
      ObjectCreate(0,"Buy",OBJ_BUTTON,0,0,0); // 创建 "买入"按钮
//--- 配置 "买入"按钮
   ObjectSetInteger(0,"Buy",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Buy",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Buy",OBJPROP_YDISTANCE,50);
   ObjectSetInteger(0,"Buy",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Buy",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Buy",OBJPROP_TEXT,"Buy");
   ObjectSetInteger(0,"Buy",OBJPROP_COLOR,clrRed);
//--- 检查名称为 "卖出"的对象
   if(ObjectFind(0,"Sell")>=0)
     {
      //--- 如果发现的对象不是按钮，请删除
      if(ObjectGetInteger(0,"Sell",OBJPROP_TYPE)!=OBJ_BUTTON)
         ObjectDelete(0,"Sell");
     }
   else
      ObjectCreate(0,"Sell",OBJ_BUTTON,0,0,0); // 创建 "卖出"按钮
//--- 配置 "卖出"按钮
   ObjectSetInteger(0,"Sell",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Sell",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Sell",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Sell",OBJPROP_TEXT,"Sell");
   ObjectSetInteger(0,"Sell",OBJPROP_COLOR,clrBlue);
//--- 执行强制更新图表，立即查看按钮
   ChartRedraw();
//---
  }
//+------------------------------------------------------------------+
//| 使用OrderSendAsync()异步函数进行买入                                |
//+------------------------------------------------------------------+
void BuyAsync(double volume)
  {
//--- 准备请求
   MqlTradeRequest req={};
   req.action      =TRADE_ACTION_DEAL;
   req.symbol      =_Symbol;
   req.magic       =MagicNumber;
   req.volume      =0.1;
   req.type        =ORDER_TYPE_BUY;
   req.price       =SymbolInfoDouble(req.symbol,SYMBOL_ASK);
   req.deviation   =10;
   req.comment     ="Buy using OrderSendAsync()";
   MqlTradeResult  res={};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
//| 使用OrderSendAsync()异步函数进行卖出                                |
//+------------------------------------------------------------------+
void SellAsync(double volume)
  {
//--- 准备请求
```

© 2000-2025, MetaQuotes Ltd.

```
   MqlTradeRequest req={};
   req.action       =TRADE_ACTION_DEAL;
   req.symbol       =_Symbol;
   req.magic        =MagicNumber;
   req.volume       =0.1;
   req.type         =ORDER_TYPE_SELL;
   req.price        =SymbolInfoDouble(req.symbol,SYMBOL_BID);
   req.deviation    =10;
   req.comment      ="Sell using OrderSendAsync()";
   MqlTradeResult  res={};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
```

在“专家”日志显示消息的示例：

© 2000-2025, MetaQuotes Ltd.

```
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnChartEvent: sparam = Sell
12:52:52   ExpertAdvisor (EURUSD,H1)   Sell EURUSD 0.01 lot
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_REQUEST
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------RequestDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_ACTION_DEAL
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Magic Number: 1234567
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order filling: ORDER_FILLING_FOK
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Deviation points: 10
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Limit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment: Sell using OrderSendAsync()
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ ResultDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   Retcode 10009
12:52:52   ExpertAdvisor (EURUSD,H1)   Request ID: 2
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Ask: 1.29319
12:52:52   ExpertAdvisor (EURUSD,H1)   Bid: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment:
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+1)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_DELETE
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
```

© 2000-2025, MetaQuotes Ltd.

```
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_HISTORY_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_FILLED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_DEAL_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
```

© 2000-2025, MetaQuotes Ltd.

```
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
```

© 2000-2025, MetaQuotes Ltd.

# PositionsTotal

返回持仓的数量。

```
int  PositionsTotal();
```

返回值

整型值

注意

对于"单边"持仓(ACCOUNT_MARGIN_MODE_RETAIL_NETTING和ACCOUNT_MARGIN_MODE_EXCHANGE),无论任何时候一个交易品种只能存在一个持仓。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓(ACCOUNT_MARGIN_MODE_RETAIL_HEDGING),那么可为一个交易品种打开多个持仓。

示例:

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得并在日志中打印账户中持仓的数量
   int total=PositionsTotal();
   Print("Number of open positions on account: ", total);
   /*
   结果:
   Number of open positions on account: 2
   */
  }
```

另见

PositionGetSymbol(), PositionSelect(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionGetSymbol

返回的交易品种与开仓位置的值保持一致，使用 PositionGetDouble, PositionGetInteger, PositionGetString函数为下一项操作自动选择位置。

```
string  PositionGetSymbol(
   int  index      // 仓位列表中的数量
   );
```

参量

*index*

[in] 开仓位置的列表位置数量。

返回值

字符串 型值，如果找不到位置，返回空字符串，为了获得 错误代码，调用 GetLastError() 函数。

注释

对于每一个交易品种，任何给出的时间段，只有一个仓位可以开放， 它能促成一笔或多笔交易的产生，不要弄混 挂单 位置，也会在客户端 "工具箱"的 "交易"中显示。

交易账户 总仓位数不能超过金融工具数 的总量。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得账户中开启的仓位数量
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- 根据 i 循环索引取得仓位交易品种
      ResetLastError();
      string symbol=PositionGetSymbol(i);

      //--- 如果成功得到交易品种, 位于 i 索引的仓位会自动被选择
      //--- 然后我们就可以使用 PositionGetDouble, PositionGetInteger 和 PositionGetString
      if(symbol!="")
        {
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         PrintFormat("Position symbol at index %d: %s, position type: %s", i, symbol,
        }
      else
        {
         PrintFormat("PositionGetSymbol(%d) failed. Error %d", i, GetLastError());
         continue;
```

© 2000-2025, MetaQuotes Ltd.

```
         }
      }
   /*
   结果:
   Position symbol at index 0: GBPUSD, position type: SELL
   Position symbol at index 1: EURUSD, position type: BUY
   */
  }
```

另见

PositionsTotal(), PositionSelect(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionSelect

为接下来的工作选择一个仓位，如果函数成功完成，返回true，如果失败返回false，调用GetLastError()函数获取错误信息。

```
bool  PositionSelect(
   string  symbol      // 交易品种名称
   );
```

参量

*交易品种*

[in] 金融证券名称

返回值

布尔型值

注释

对于每个交易品种，任何给出时间，仅可以持有一个仓位，它是一个或者多个交易的结果，不要弄混挂单仓位，也会在客户端 "工具箱"的 "交易"中显示。

函数PositionSelect()复制程序函数的位置数据，进一步调用PositionGetDouble(), PositionGetInteger() 和 PositionGetString() 返回较早的复制数据。表明该位置本身已经不再存在（或者成交量，方向等的改变），但位置数据仍旧可以获得，为了确保关于位置的新数据的接受，在提及他们之前，推荐使用PositionSelect()函数。

示例：

```
#define   SYMBOL_NAME   "EURUSD"

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 选择指定交易品种的仓位
   if(!PositionSelect(SYMBOL_NAME))
     {
      PrintFormat("PositionSelect(%s) failed. Error %d",SYMBOL_NAME, GetLastError());
      return;
     }

//--- 如果选定了仓位, 我们可以使用 PositionGetDouble(), PositionGetInteger() 和 PositionGe
//--- 取得选定仓位的编号
   ResetLastError();
   long ticket=PositionGetInteger(POSITION_TICKET);
   if(ticket==0)
     {
      PrintFormat("Failed to get %s position ticket. Error %d", SYMBOL_NAME, GetLastEr
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 如果成功取得编号，在日志中打印选定仓位的交易品种和编号
   PrintFormat("The position that is selected on the %s symbol has ticket %I64d", SYMB
   /*
   结果：
   The position that is selected on the EURUSD symbol has ticket 2810846623
   */
  }
```

另见

PositionGetSymbol(), PositionsTotal(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionSelectByTicket

根据持仓中指定的报价号选择持仓。如果成功，返回true。如果函数失败，返回false。调用GetLastError()了解错误详情。

```
bool  PositionSelectByTicket(
   ulong   ticket     // 持仓价格
   );
```

参数

*报价*

[in] 持仓报价。

返回值

bool型值。

注意

PositionSelectByTicket() 函数复制持仓数据到程序环境。进一步调用PositionGetDouble(), PositionGetInteger() 和 PositionGetString() 返回之前复制的数据。即使持仓已不存在 (或其大小，方向等等发生变化)，有时仍然可能接收到数据。若要确保您收到有效的持仓数据，建议您访问数据之前调用PositionSelectByTicket()。

示例：

```
#define   EXPERT_MAGIC  123456   // MagicNumber

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 声明和初始化请求和结果结构
   MqlTradeRequest request={};
   MqlTradeResult  result ={};

//--- 填充交易请求参数以开启一个多头仓位
   request.action    = TRADE_ACTION_DEAL;                       // 交易操作类型
   request.symbol    = Symbol();                                // 交易品种
   request.volume    = 0.1;                                     // 交易量为 0.1 手
   request.type      = ORDER_TYPE_BUY;                          // 订单类型
   request.price     = SymbolInfoDouble(Symbol(), SYMBOL_ASK); // 开仓价格
   request.deviation = 5;                                       // 允许的价格偏差
   request.magic     = EXPERT_MAGIC;                            // 订单幻数

//--- 发送请求. 如果发送请求失败, 则显示错误代码并结束操作
   if(!OrderSend(request, result))
     {
      PrintFormat("OrderSend error ", GetLastError());
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 显示操作数据
   PrintFormat("Trade request result: retcode: %u, deal: %I64u, order: %I64u", result.

//--- 取得交易操作结果的仓位编号并根据编号选择仓位
//--- 新开启仓位的编号就对应着所生成交易的订单编号
   ulong ticket=result.order;
   ResetLastError();
   if(!PositionSelectByTicket(ticket))
     {
      PrintFormat("PositionSelectByTicket(%I64u) failed. Error %d", ticket, GetLastErr
      return;
     }

//--- 在日志中根据选定编号显示仓位数据
   ENUM_POSITION_TYPE type   = (ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
   long               time   = PositionGetInteger(POSITION_TIME_MSC);
   double             price  = PositionGetDouble(POSITION_PRICE_OPEN);
   double             volume= PositionGetDouble(POSITION_VOLUME);
   string             symbol= PositionGetString(POSITION_SYMBOL);
   int                digits= (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   PrintFormat("Current selected position: %s %.2f %s #%I64u at %.*f, %s",
               symbol, volume, (type==POSITION_TYPE_BUY ? "Buy" : "Sell"), ticket, dig
   /*
   结果:
   Trade request result: retcode: 10009, deal: 2778100901, order: 2803905975
   Current selected position: EURUSD 0.10 Buy #2803905975 at 1.10672, 2024.09.02 12:09
   */
  }
//+------------------------------------------------------------------+
//| 返回包含毫秒信息的时间                                           |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 3
  }
```

另见

PositionGetSymbol(), PositionsTotal(), 持仓属性

© 2000-2025, MetaQuotes Ltd.

# PositionGetDouble

函数返回持仓的要求性质，使用PositionGetSymbol或者 PositionSelect预选出来，仓位属性必须是双精度型，有两种变体函数。

1. 立即返回属性值。

```
double  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id      // 属性标识符
   );
```

2. 返回true或者false，取决于函数执行成功与否，如果成功，性质值通过引用安置在最后参量的接受变量里。

```
bool  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id,     // 属性标识符
   double&                        double_var       // 这里接受属性值
   );
```

参量

*property_id*

[in]位置属性标识符。值可能是ENUM_POSITION_PROPERTY_DOUBLE 计算中一个。

*double_var*

[out] 双精度类型值，如果函数调用失败，返回0。

返回值

双精度 类型值。如果函数失败，返回0 。

注释

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

为确保关于仓位的新数据的接收，在提及之前调用 PositionSelect() 函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中的所有仓位中循环
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 通过自动选择仓位取得下一个仓位的编号以访问它的属性
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得仓位类型并显示仓位实数属性列表的标题
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("Double properties of an open position %s #%I64u:", type, ticket);

      //--- 在标题下方打印选定仓位的全部实数属性
      PositionPropertiesDoublePrint(15);
     }
   /*
   结果:
   Double properties of an open position Buy #2807075208:
   Volume:        1.00
   Price open:    1.10516
   StopLoss:      0.00000
   TakeProfit:    0.00000
   Price current: 1.10518
   Swap:          0.00
   Profit:        2.00 USD
   */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选定仓位的实数属性                                   |
//+------------------------------------------------------------------+
void PositionPropertiesDoublePrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;

//--- 取得账户货币, 仓位的交易品种和交易品种的小数点位数
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   string symbol  =PositionGetString(POSITION_SYMBOL);
   int    digits  =(int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0, 则宽度为标题文字行的大小+1
   header="Volume:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中以指定的标题宽度显示仓位交易量
   if(!PositionGetDouble(POSITION_VOLUME, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示仓位价格值
   header="Price open:";
```

© 2000-2025, MetaQuotes Ltd.

```
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PRICE_OPEN, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示止损值
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示止盈值
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示当前价格值
   header="Price current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PRICE_CURRENT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示累积的库存费
   header="Swap:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_SWAP, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示当前利润值
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PROFIT, value))
      return;
   PrintFormat("%-*s%-.2f %s", w, header, value, currency);
  }
```

另见

PositionGetSymbol(), PositionSelect(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionGetInteger

函数返回持仓需求属性，使用 PositionGetSymbol 或者 PositionSelect. 预选出来，仓位属性的类型是时间日期整型，有两种变体函数。

1. 立即返回属性值

```
long  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id      // 属性标识符
   );
```

2. 返回true或者false，取决于函数执行成功与否，如果成功，性质值通过引用安置在最后参量的接受变量里。

```
bool  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id,     // 属性标识符
   long&                           long_var         // 这里接受属性值
   );
```

参量

*property_id*

[in] 仓位属性标识符，值可能是 ENUM_POSITION_PROPERTY_INTEGER 中的一个。

*long_var*

[out] 接受要求性质的字符串变量值。

返回值

长整型 值。如果函数失败，返回0。如果函数失败，返回0。

注释

对于"单边"持仓的解释就是 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE)，在任何时候一个交易品种 只能存在一个持仓。该持仓是一个或多个交易的结果。不要混淆有效 挂单的持仓，它只显示在工具箱窗口的交易标签。

如果允许单个持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么一个交易品种可以打开多个持仓。

为确保关于仓位的新数据的接收，在提及之前调用 PositionSelect() 函数。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 交易函数                                                         |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- 检查持仓是否存在并显示其变化的时间
   if(PositionSelect(_Symbol))
     {
//--- 接收持仓ID为进一步工作
      ulong position_ID=PositionGetInteger(POSITION_IDENTIFIER);
      Print(_Symbol," position #",position_ID);
//--- 接收从1970.01.01开始的毫秒为单位的形成持仓的时间
      long create_time_msc=PositionGetInteger(POSITION_TIME_MSC);
      PrintFormat("Position #%d  POSITION_TIME_MSC = %i64 milliseconds => %s",positior
                  create_time_msc,TimeToString(create_time_msc/1000));
//--- 接收从1970.01.01开始的秒为单位的持仓最近变化的时间
      long update_time_sec=PositionGetInteger(POSITION_TIME_UPDATE);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE = %i64 seconds => %s",
                  position_ID,update_time_sec,TimeToString(update_time_sec));
//--- 接收从1970.01.01开始的毫秒为单位的持仓最近变化的时间
      long update_time_msc=PositionGetInteger(POSITION_TIME_UPDATE_MSC);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE_MSC = %i64 milliseconds => %s",
                  position_ID,update_time_msc,TimeToString(update_time_msc/1000));
     }
//---
  }
```

另见

PositionGetSymbol(), PositionSelect(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionGetString

函数返回持仓需求属性，使用PositionGetSymbol或者PositionSelect预选出来。仓位属性可以是字符串型。有2种变体函数。

1. 立即返回属性值

```
string  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id      // 属性标识符
   );
```

2. 返回true或者false，取决于函数执行成功与否，如果成功，性质值通过引用安置在最后参量的接受变量里。

```
bool  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id,     // 属性标识符
   string&                        string_var       // 这里接受属性值
   );
```

参量

*property_id*

[in] 仓位属性标识符，值可能是 ENUM_POSITION_PROPERTY_STRING 中的一个。

*string_var*

[out] 接受要求性质的字符串变量值。

返回值

字符串 值。如果函数失败，返回空字符串。

注释

对于"单边"持仓的解释就是 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE)，在任何时候一个交易品种 只能存在一个持仓。 该持仓是一个或多个交易的结果。不要混淆有效 挂单的持仓，它只显示在工具箱窗口的交易标签。

如果允许单个持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么一个交易品种可以打开多个持仓。

为确保关于仓位的新数据的接收，在提及之前调用 PositionSelect() 函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中的所有仓位中循环
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- 通过自动选择仓位取得下一个仓位的编号以访问它的属性
```

© 2000-2025, MetaQuotes Ltd.

```
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得仓位类型并显示用于仓位字符串属性列表的标题
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("String properties of an open position %s #%I64u:", type, ticket);

      //--- 在标题下方打印选定仓位的所有字符串型属性
      PositionPropertiesStringPrint(15);
     }
   /*
  结果:
   String properties of an open position Buy #2810798881:
   Symbol:         EURUSD
   Comment:        Test PositionGetString
   External ID:
   */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选定仓位的字符串型属性 |
//+------------------------------------------------------------------+
void PositionPropertiesStringPrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value="";

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0, 则宽度为标题文字行的大小+1
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中以指定标题宽度显示仓位的交易品种
   if(!PositionGetString(POSITION_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 在日志中显示仓位注释
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetString(POSITION_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 在日志中显示外部系统中的仓位 ID
   header="External ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetString(POSITION_EXTERNAL_ID, value))
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
   PrintFormat("%-*s%-s", w, header, value);
  }
```

另见

PositionGetSymbol(), PositionSelect(), 仓位属性

© 2000-2025, MetaQuotes Ltd.

# PositionGetTicket

该函数返回持仓列表中指定索引的持仓报价，自动选择使用 PositionGetDouble, PositionGetInteger, PositionGetString函数的持仓。

```
ulong  PositionGetTicket(
   int  index      // 列表中的持仓编号
   );
```

参数

*index*

[in] 持仓列表中的持仓索引，编号从0开始。

返回值

持仓报价。如果函数失败，返回0。

注意

对于"单边"持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

若要确保收到持仓的最新数据，建议查阅之前直接调用 PositionSelect()函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中的所有仓位中循环
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- 通过自动选择仓位取得下一个仓位的编号以访问它的属性
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得仓位类型并在日志中显示选定仓位的描述
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("[%d] Selected position %s #%I64u", i, type, ticket);
     }
   /*
   结果：
   [0] Selected position Sell #2810802718
```

© 2000-2025, MetaQuotes Ltd.

```
   [1] Selected position Buy #2810802919
   */
  }
```

另见

PositionGetSymbol(), PositionSelect(), Position Properties

© 2000-2025, MetaQuotes Ltd.

# OrdersTotal

返回当前订单数。

```
int  OrdersTotal();
```

返回值

整型值。

注释

不要使仓位挂单混乱，它显示在客户端“工具箱”的“交易”标签中。订单是交易要求，而仓位就是一个或者多个交易的结果。

对于"单边"持仓(ACCOUNT_MARGIN_MODE_RETAIL_NETTING和ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓(ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 取得并在日志中打印账户中活动的挂单数量
   int total=OrdersTotal();
   Print("Number of active pending orders on the account: ", total);
   /*
   结果:
   Number of active pending orders on the account: 2
   */
  }
```

另见

OrderSelect(), OrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# OrderGetTicket

返回类似订单票据，使用函数自动选择订单工作.

```
ulong  OrderGetTicket(
   int  index      // 订单列表中的数量
   );
```

参量

*index*

[in] 当前订单列表中的订单数量。

返回值

无符长整型 值。如果函数失败，返回0。

注释

不要使仓位挂单混乱，它显示在客户端 "工具箱"的 "交易"标签中。订单是交易要求，而仓位就是一个或者多个交易的结果。

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

OrderGetTicket()函数复制有关订单的数据到程序环境中，远程调用OrderGetDouble(), OrderGetInteger(), OrderGetString()，返回先前的复制数据。表示该命令本身不存在 (或者开仓价格，止损数值/目标数值水平或者届期改变)，但是命令数据仍旧包含。为了确保接收订单新命令，推荐调用OrderGetTicket()命令。

示例：

```
void OnStart()
  {
//--- 订单属性返回值的变量
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   string   symbol;
   string   type;
   long     order_magic;
//--- 当前挂单量
   uint     total=OrdersTotal();
//--- 反复检查通过订单
   for(uint i=0;i<total;i++)
     {
      //--- 通过列表中的仓位返回订单报价
```

© 2000-2025, MetaQuotes Ltd.

```
      if(ticket=OrderGetTicket(i))
        {
         //--- 返回订单属性
         open_price    =OrderGetDouble(ORDER_PRICE_OPEN);
         time_setup    =(datetime)OrderGetInteger(ORDER_TIME_SETUP);
         symbol        =OrderGetString(ORDER_SYMBOL);
         order_magic   =OrderGetInteger(ORDER_MAGIC);
         positionID    =OrderGetInteger(ORDER_POSITION_ID);
         initial_volume=OrderGetDouble(ORDER_VOLUME_INITIAL);
         type          =EnumToString(ENUM_ORDER_TYPE(OrderGetInteger(ORDER_TYPE)));
         //--- 准备和显示订单信息
         printf("#ticket %d %s %G %s at %G was set up at %s",
                ticket,                  // 订单报价
                type,                    // 类型
                initial_volume,          // 已下交易量
                symbol,                  // 交易品种
                open_price,              // 规定的开盘价
                TimeToString(time_setup)// 下订单时间
                );
        }
     }
//---
  }
```

另见

OrdersTotal(), OrderSelect(), OrderGetInteger()

© 2000-2025, MetaQuotes Ltd.

# OrderSelect

选择工作订单。如果函数成功调用返回真值，如果函数没能完成，返回错误值，了解更多关于错误调用的信息，使用 GetLastError()。

```
bool  OrderSelect(
   ulong    ticket      // 订单号
   );
```

参量

*报价*

[in] 订单号。

返回值

布尔型值。

注释

不要使仓位挂单混乱，它显示在客户端 "工具箱"的 "交易"标签中。

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

OrderSelect()函数复制有关订单的数据到程序环境中，远程调用OrderGetDouble(), OrderGetInteger(), OrderGetString()，返回先前的复制数据。表示该命令本身不存在 (或者开仓价格，止损数值/目标数值水平或者届期改变)，但是命令数据仍旧包含。为了确保接收订单新命令，推荐调用OrderSelect()命令。

示例：

```
#define   EXPERT_MAGIC  123456
#define   OFFSET        50                      // 设置订单距离当前价格的点数
#define   DIRECTION     ORDER_TYPE_BUY_LIMIT    // 订单类型
#define   VOLUME        1.0                     // 交易量
#define   DEVIATION     2                       // 允许的价格偏差

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//-- 声明和初始化交易请求和结果以及变量
   MqlTradeRequest request={};
   MqlTradeResult  result ={};
   double          order_price=0;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置挂单的参数
   request.action    = TRADE_ACTION_PENDING;                                  // 交易操作类
   request.symbol    = _Symbol;                                               // 交易品种
   request.volume    = VOLUME;                                                // 交易量
   request.deviation = DEVIATION;                                             // 允许的价格
   request.magic     = EXPERT_MAGIC;                                          // 订单幻数

//--- 检查操作类型
   switch(DIRECTION)
     {
      case ORDER_TYPE_BUY_LIMIT :
        request.type = ORDER_TYPE_BUY_LIMIT;                                  // 订单类型
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_ASK)-OFFSET*_Point;// 开单价格
        request.price= NormalizeDouble(order_price, _Digits);                // 规范化开单
        break;
      case ORDER_TYPE_SELL_LIMIT :
        request.type = ORDER_TYPE_SELL_LIMIT;                                 // 订单类型
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_BID)+OFFSET*_Point;// 开单价格
        request.price= NormalizeDouble(order_price,_Digits);                 // 规范化开单
        break;
      case ORDER_TYPE_BUY_STOP :
        request.type = ORDER_TYPE_BUY_STOP;                                   // 订单类型
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_ASK)+OFFSET*_Point;// 开单价格
        request.price= NormalizeDouble(order_price,_Digits);                 // 规范化开单
        break;
      case ORDER_TYPE_SELL_STOP :
        request.type = ORDER_TYPE_SELL_STOP;                                  // 订单类型
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_BID)-OFFSET*_Point;// 开单价格
        request.price= NormalizeDouble(order_price,_Digits);                 // 规范化开单
        break;
      default: // 如果没有选择挂单或者止损限价单
        Alert("This example is only for placing pending orders BuyLimit, SellLimit, Bu
        break;
     }

//--- 发送请求. 如果发送请求失败, 则显示错误代码并结束操作
   if(!OrderSend(request, result))
     {
      Print("OrderSend error ", GetLastError());
      return;
     }

//--- 显示操作数据
   PrintFormat("Trade request result: retcode=%u, order=%I64u", result.retcode, result

//--- 从交易操作结果中取得订单编号并根据编号选择订单
   ulong ticket=result.order;
   ResetLastError();
   if(!OrderSelect(ticket))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       PrintFormat("OrderSelect(%I64u) failed. Error %d", ticket, GetLastError());
       return;
      }

//--- 在日志中显示根据编号选中的订单数据
   ENUM_ORDER_TYPE    type   = (ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE);
   long               time   = OrderGetInteger(ORDER_TIME_SETUP_MSC);
   double             price = OrderGetDouble(ORDER_PRICE_OPEN);
   double             volume= OrderGetDouble(ORDER_VOLUME_CURRENT);
   string             symbol= OrderGetString(ORDER_SYMBOL);
   int                digits= (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   PrintFormat("Current selected order: %s %.2f %s #%I64u at %.*f, %s",
               symbol, volume, OrderTypeDescription(type), ticket, digits, price, Time
   /*
   结果:
   Trade request result: retcode=10009, order=2811006719
   Current selected order: EURUSD 1.00 Buy Limit #2811006719 at 1.10550, 2024.09.04 1
   */
  }
//+------------------------------------------------------------------+
//| 返回包含毫秒信息的时间                                           |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
```

另见

OrderGetInteger(), OrderGetDouble(), OrderGetString(), OrderCalcProfit(), OrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# OrderGetDouble

返回订单的需求属性，使用 OrderGetTicket 或者 OrderSelect 进行预选择。订单属性必须是双精度类型。有两种变量函数可以使用。

1. 立即返回属性值。

```
double  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id        // 属性标识符
   );
```

2. 依据函数是否成功调用返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id,       // 属性标识符
   double&                     double_var         // 这里接受属性值
   );
```

参量

*property_id*

[in] 订单性质标识符，值可以是 ENUM_ORDER_PROPERTY_DOUBLE 值中的一个。

*double_var*

[out] 双精度类型变量接收要求属性的值。

返回值

双精度 值。如果函数失败，返回0。

注释

不要使仓位挂单混乱，它显示在客户端 "工具箱"的 "交易"标签中。

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

为确保关于新订单的新数据的接受，推荐调用 OrderSelect() 函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中所有订单的列表中做循环
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
```

© 2000-2025, MetaQuotes Ltd.

```
   {
    //--- 根据循环索引取得列表中的订单编号
    ulong ticket=OrderGetTicket(i);
    if(ticket==0)
       continue;

    //--- 取得订单类型并显示用于选中订单实数型属性列表的标题
    string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
    PrintFormat("Double properties of an active pending order %s #%I64u:", type, tic

    //--- 在标题下方打印选中订单的全部实数型属性
    OrderPropertiesDoublePrint(16);
   }
/*
结果:
Double properties of an active pending order Sell Limit #2812000714:
Volume initial: 1.00
Volume current: 1.00
Price open:     145.282
StopLoss:       0.000
TakeProfit:     0.000
Price current:  145.044
StopLimit:      0.000
Double properties of an active pending order Buy Limit #2812001112:
Volume initial: 1.00
Volume current: 1.00
Price open:     144.836
StopLoss:       0.000
TakeProfit:     0.000
Price current:  145.051
StopLimit:      0.000
Double properties of an active pending order Buy Stop #2812001488:
Volume initial: 0.50
Volume current: 0.50
Price open:     1.10642
StopLoss:       0.00000
TakeProfit:     0.00000
Price current:  1.10530
StopLimit:      0.00000
Double properties of an active pending order Sell Stop #2812001712:
Volume initial: 0.50
Volume current: 0.50
Price open:     1.10374
StopLoss:       0.00000
TakeProfit:     0.00000
Price current:  1.10525
StopLimit:      0.00000
*/
}
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
//+------------------------------------------------------------------+
//| 在日志中显示选定订单的实数型属性                                 |
//+------------------------------------------------------------------+
void OrderPropertiesDoublePrint(const uint header_width=0)
  {
//--- 取得订单的交易品种和交易品种的小数位数
   string symbol = OrderGetString(ORDER_SYMBOL);
   int    digits = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- 在日志中以指定宽度的标题显示下单时的初始交易量
   OrderPropertyPrint("Volume initial:",header_width,2,ORDER_VOLUME_INITIAL);

//--- 在日志中显示未执行的订单交易量
   OrderPropertyPrint("Volume current:",header_width,2,ORDER_VOLUME_CURRENT);

//--- 在日志中显示订单中指定的价格
   OrderPropertyPrint("Price open:",header_width,digits,ORDER_PRICE_OPEN);

//--- 在日志中显示止损水平
   OrderPropertyPrint("StopLoss:",header_width,digits,ORDER_SL);

//--- 在日志中显示止盈水平
   OrderPropertyPrint("TakeProfit:",header_width,digits,ORDER_TP);

//--- 在日志中显示订单交易品种的当前价格
   OrderPropertyPrint("Price current:",header_width,digits,ORDER_PRICE_CURRENT);

//--- 在日志中显示限价止损订单的订单限价
   OrderPropertyPrint("StopLimit:",header_width,digits,ORDER_PRICE_STOPLIMIT);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 在日志中显示订单实数型属性值                                                |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, int digits, ENUM_ORDE
  {
   double value=0;
   if(!OrderGetDouble(property, value))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLastE
   else
     {
      //--- 如果传递给函数的标题宽度为0，则宽度为标题行大小+1
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-.*f", w, header, digits, value);
     }
  }
```

另见

OrdersTotal(), OrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# OrderGetInteger

返回订单性质的要求，使用OrderGetTicket 或者 OrderSelect重新选择，属性一定是日期时间，整型。有2种变量函数可以使用。

1. 立即返回属性值。

```
long  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id         // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id,       // 属性标识符
   long&                        long_var           // 这里接受属性值
   );
```

参量

*property_id*

[in] 订单性质标识符。值可以是 ENUM_ORDER_PROPERTY_INTEGER 列举值中的一个。

*long_var*

[out] 接收要求属性值的长整型变量。

返回值

长整型 值。如果函数失败，返回0。

注释

不要使仓位挂单混乱，它显示在客户端 "工具箱"的 "交易"标签中。

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

为确保关于新订单的新数据的接受，推荐调用 OrderSelect() 函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中所有订单的列表中做循环
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 根据循环索引取得列表中的订单编号
       ulong ticket=OrderGetTicket(i);
       if(ticket==0)
          continue;

       //--- 取得订单类型并显示用于选中订单实数型属性列表的标题
       string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
       PrintFormat("Integer properties of an active pending order %s #%I64u:", type, ti

       //--- 在标题下打印选定订单的所有整数型属性
       OrderPropertiesIntegerPrint(17);
      }
    /*
    结果:
    Integer properties of an active pending order Buy Limit #2812945317:
    Ticket:           2812945317
    Time setup:       2024.09.04 19:17:16
    Type:             Buy Limit
    State:            Placed
    Time expiration:  0
    Time done:        0
    Time setup msc:   2024.09.04 19:17:16.686
    Time done msc:    0
    Type filling:     Return
    Type time:        Time GTC
    Magic:            0
    Reason:           Client
    Position ID:      0
    Position By ID:   0
    */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选定订单的整数型属性                                 |
//+------------------------------------------------------------------+
void OrderPropertiesIntegerPrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   long   value=0;

//--- 在日志中显示订单编号
   OrderPropertyPrint("Ticket:", header_width, ORDER_TICKET);

//--- 在日志中显示订单下单时间
   OrderPropertyPrint("Time setup:", header_width, ORDER_TIME_SETUP);

//--- 在日志中显示订单类型
   OrderPropertyPrint("Type:", header_width, ORDER_TYPE);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在日志中显示订单状态
   OrderPropertyPrint("State:", header_width, ORDER_STATE);

//--- 在日志中显示订单过期时间
   OrderPropertyPrint("Time expiration:", header_width, ORDER_TIME_EXPIRATION);

//--- 在日志中显示订单执行/过期时间
   OrderPropertyPrint("Time done:", header_width, ORDER_TIME_DONE);

//--- 在日志中显示订单设置的时间, 以自1970年1月1日开始计算的毫秒数为单位
   OrderPropertyPrint("Time setup msc:", header_width, ORDER_TIME_SETUP_MSC);

//--- 在日志中显示订单执行/过期的时间, 以自1970年1月1日开始计算的毫秒数为单位1
   OrderPropertyPrint("Time done msc:", header_width, ORDER_TIME_DONE_MSC);

//--- 在日志中显示订单的执行类型1
   OrderPropertyPrint("Type filling:", header_width, ORDER_TYPE_FILLING);

//--- 在日志中显示订单的生命周期
   OrderPropertyPrint("Type time:", header_width, ORDER_TYPE_TIME);

//--- 在日志中显示下订单的 EA 的 ID
   OrderPropertyPrint("Magic:", header_width, ORDER_MAGIC);

//--- 在日志中显示订单原因或来源
   OrderPropertyPrint("Reason:", header_width, ORDER_REASON);

//--- 在日志中显示订单执行过程中的仓位 ID
   OrderPropertyPrint("Position ID:", header_width, ORDER_POSITION_ID);

//--- 在日志中显示 ORDER_TYPE_CLOSE_BY 类型订单的反向仓位 ID
   OrderPropertyPrint("Position By ID:", header_width, ORDER_POSITION_BY_ID);
  }
//+------------------------------------------------------------------+
//| 在日志中显示订单的整数型属性值                                   |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, ENUM_ORDER_PROPERTY_IN
  {
   string svalue="";
   long   lvalue=0;
   if(!OrderGetInteger(property, lvalue))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLastE
   else
     {
      switch(property)
        {
         case ORDER_TICKET          :
         case ORDER_MAGIC           :
```

© 2000-2025, MetaQuotes Ltd.

```
         case ORDER_POSITION_ID     :
         case ORDER_POSITION_BY_ID  :
           svalue=(string)lvalue;
           break;

         case ORDER_TIME_SETUP      :
         case ORDER_TIME_EXPIRATION :
         case ORDER_TIME_DONE       :
           svalue=(lvalue!=0 ? TimeToString((datetime)lvalue, TIME_DATE|TIME_MINUTES|T
           break;

         case ORDER_TIME_SETUP_MSC  :
         case ORDER_TIME_DONE_MSC   :
           svalue=(lvalue!=0 ? TimeMscToString(lvalue) : "0");
           break;

         case ORDER_TYPE            :
           svalue=OrderTypeDescription((ENUM_ORDER_TYPE)lvalue);
           break;
         case ORDER_STATE           :
           svalue=OrderStateDescription((ENUM_ORDER_STATE)lvalue);
           break;
         case ORDER_TYPE_FILLING    :
           svalue=OrderTypeFillingDescription((ENUM_ORDER_TYPE_FILLING)lvalue);
           break;
         case ORDER_TYPE_TIME       :
           svalue=OrderTypeTimeDescription((ENUM_ORDER_TYPE_TIME)lvalue);
           break;
         case ORDER_REASON          :
           svalue=OrderReasonDescription((ENUM_ORDER_REASON)lvalue);
           break;

         default                    :
           svalue="Unknown property";
           break;
        }

      //--- 如果传递给函数的标题宽度为0, 则宽度为标题行大小+1
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-s", w, header, svalue);
     }
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回订单状态的描述                                               |
//+------------------------------------------------------------------+
string OrderStateDescription(ENUM_ORDER_STATE state)
  {
   switch(state)
     {
      case ORDER_STATE_STARTED         :  return("Started");
      case ORDER_STATE_PLACED          :  return("Placed");
      case ORDER_STATE_CANCELED        :  return("Canceled");
      case ORDER_STATE_PARTIAL         :  return("Partial");
      case ORDER_STATE_FILLED          :  return("Filled");
      case ORDER_STATE_REJECTED        :  return("Rejected");
      case ORDER_STATE_EXPIRED         :  return("Expired");
      case ORDER_STATE_REQUEST_ADD     :  return("Request Add");
      case ORDER_STATE_REQUEST_MODIFY  :  return("Request Modify");
      case ORDER_STATE_REQUEST_CANCEL  :  return("Request Cancel");
      default                          :  return("Unknown state: "+(string)state);
     }
  }
//+------------------------------------------------------------------+
//| 返回订单交易量执行政策的描述                                     |
//+------------------------------------------------------------------+
string OrderTypeFillingDescription(const ENUM_ORDER_TYPE_FILLING type)
  {
   switch(type)
     {
      case ORDER_FILLING_FOK     :  return("Fill or Kill");
      case ORDER_FILLING_IOC     :  return("Immediate or Cancel");
      case ORDER_FILLING_BOC     :  return("Book or Cancel");
      case ORDER_FILLING_RETURN  :  return("Return");
      default                    :  return("Unknown type filling: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回订单过期时间的描述                                           |
//+------------------------------------------------------------------+
string OrderTypeTimeDescription(const ENUM_ORDER_TYPE_TIME type)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   switch(type)
     {
      case ORDER_TIME_GTC           :   return("Time GTC");
      case ORDER_TIME_DAY           :   return("Time Day");
      case ORDER_TIME_SPECIFIED     :   return("Time Specified");
      case ORDER_TIME_SPECIFIED_DAY :   return("Time Specified Day");
      default                       :  return("Unknown type time: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回下单原因的描述                                               |
//+------------------------------------------------------------------+
string OrderReasonDescription(const ENUM_ORDER_REASON reason)
  {
   switch(reason)
     {
      case ORDER_REASON_CLIENT    :  return("Client");
      case ORDER_REASON_MOBILE    :  return("Mobile");
      case ORDER_REASON_WEB       :  return("Web");
      case ORDER_REASON_EXPERT    :  return("Expert");
      case ORDER_REASON_SL        :  return("Stop Loss");
      case ORDER_REASON_TP        :  return("Take Profit");
      case ORDER_REASON_SO        :  return("Stop Out");
      default                     :  return("Unknown reason: "+(string)reason);
     }
  }
//+------------------------------------------------------------------+
//| 返回包含毫秒信息的时间                                           |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 3
  }
```

另见

OrdersTotal(), OrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# OrderGetString

返回要求的订单属性，使用 OrderGetTicket 或者 OrderSelect预选择。订单属性一定是字符串型。有2种变量函数可以使用。

1. 立即返回属性值。

```
string  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id,       // 属性标识符
   string&                     string_var        // 这里接受属性值
   );
```

参量

*property_id*

[in] 订单性质标识符，值可以是 ENUM_ORDER_PROPERTY_STRING 列举值中的一个。

*string_var*

[out] 接收要求属性值的字符串型变量。

返回值

字符串 类型。

注释

不要使仓位挂单混乱，它显示在客户端 “工具箱”的 “交易”标签中。

对于"单边" 持仓 (ACCOUNT_MARGIN_MODE_RETAIL_NETTING 和 ACCOUNT_MARGIN_MODE_EXCHANGE)，无论任何时候一个交易品种只能存在一个持仓 。该持仓是一个或多个交易的结果。请不要混淆也显示在工具箱窗口交易标签中的有效挂单的持仓。

如果允许单独持仓 (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)，那么可为一个交易品种打开多个持仓。

为确保关于新订单的新数据的接受，推荐调用 OrderSelect() 函数。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在账户中所有订单的列表中做循环
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 根据循环索引取得列表中的订单编号
       ulong ticket=OrderGetTicket(i);
       if(ticket==0)
          continue;

       //--- 取得订单类型并显示用于选中订单字符串型属性列表的标题
       string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
       PrintFormat("String properties of an active pending order %s #%I64u:", type, tic

       //--- 在标题下方打印选中订单的所有字符串型属性
       OrderPropertiesStringPrint(13);
      }
    /*
    结果:
    String properties of an active pending order Sell Limit #2813781342:
    Comment:      Test OrderGetString
    Symbol:       EURUSD
    External ID:
    */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选定订单的字符串型属性                                |
//+------------------------------------------------------------------+
void OrderPropertiesStringPrint(const uint header_width=0)
  {
//--- 在日志中显示注释
   OrderPropertyPrint("Comment:", header_width, ORDER_COMMENT);

//--- 在日志中显示所下订单的交易品种
   OrderPropertyPrint("Symbol:", header_width, ORDER_SYMBOL);

//--- 在日志中显示在外部系统中的订单 ID
   OrderPropertyPrint("External ID:", header_width, ORDER_EXTERNAL_ID);
  }
//+------------------------------------------------------------------+
//| 在日志中显示订单字符串属性值                                      |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, ENUM_ORDER_PROPERTY_ST
  {
   string value="";
   if(!OrderGetString(property, value))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLastE
   else
     {
      //--- 如果传递给函数的标题宽度为0, 则宽度为标题行大小+1
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-s", w, header, value);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
```

另见

OrdersTotal(), OrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistorySelect

在服务器特定时间段，检索交易和订单的历史。

```
bool  HistorySelect(
   datetime  from_date,     // 开始日期
   datetime  to_date        // 结束日期
   );
```

参量

*from_date*

[in] 要求开始日期。

*to_date*

[in] 要求的结束日期。

返回值

如果成功返回true，否则返回false。

注释

HistorySelect()建立一系列MQL5程序的订单和交易列表，使用类似函数来执行列表元素。订单列表大小可以使用 HistoryDealsTotal() 函数返回。使用 HistoryOrdersTotal()可以显示历史订单列表的大小。在订单列表中选择应该使用 HistoryOrderGetTicket()，订单列表 HistoryDealGetTicket() 更适合。

在使用 HistoryOrderSelect()后，MQL5程序中的历史可用订单列表就会重设并被发现的订单填满。如果成功执行 the search of an order by the ticket，同样适用于MQL5程序可行性订单中-通过 HistoryDealSelect() 重设并再次成功填满。

示例：

```
void OnStart()
  {
   color BuyColor =clrBlue;
   color SellColor=clrRed;
//--- 请求交易历史记录
   HistorySelect(0,TimeCurrent());
//--- 创建物件
   string   name;
   uint     total=HistoryDealsTotal();
   ulong    ticket=0;
   double   price;
   double   profit;
   datetime time;
   string   symbol;
   long     type;
   long     entry;
//--- 所有交易
```

© 2000-2025, MetaQuotes Ltd.

```
   for(uint i=0;i<total;i++)
     {
      //--- 尽力获得交易报价
      if((ticket=HistoryDealGetTicket(i))>0)
        {
         //--- 获得交易属性
         price =HistoryDealGetDouble(ticket,DEAL_PRICE);
         time  =HistoryDealGetInteger(ticket,DEAL_TIME);
         symbol=HistoryDealGetString(ticket,DEAL_SYMBOL);
         type  =HistoryDealGetInteger(ticket,DEAL_TYPE);
         entry =HistoryDealGetInteger(ticket,DEAL_ENTRY);
         profit=HistoryDealGetDouble(ticket,DEAL_PROFIT);
         //--- 只对当前交易品种
         if(price && time && symbol==Symbol())
           {
            //--- 创建价格物件
            name="TradeHistory_Deal_"+string(ticket);
            if(entry) ObjectCreate(0,name,OBJ_ARROW_RIGHT_PRICE,0,time,price,0,0);
            else      ObjectCreate(0,name,OBJ_ARROW_LEFT_PRICE,0,time,price,0,0);
            //--- 设置物件属性
            ObjectSetInteger(0,name,OBJPROP_SELECTABLE,0);
            ObjectSetInteger(0,name,OBJPROP_BACK,0);
            ObjectSetInteger(0,name,OBJPROP_COLOR,type?BuyColor:SellColor);
            if(profit!=0) ObjectSetString(0,name,OBJPROP_TEXT,"Profit: "+string(profit
           }
        }
     }
//--- 应用于图表
   ChartRedraw();
  }
```

另见

HistoryOrderSelect(), HistoryDealSelect()

© 2000-2025, MetaQuotes Ltd.

# HistorySelectByPosition

检索特殊位置标识符交易和订单历史记录。

```
bool  HistorySelectByPosition(
   ulong   position_id     // 仓位标识符 - POSITION_IDENTIFIER
   );
```

参量

*position_id*

[in] 位置标识符设置每个已执行命令和每笔交易。

返回值

如果成功返回真值，否则返回错误值。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

HistorySelectByPosition()建立一系列命令列表和MQL5程序的交易，使用类似 仓位标识符 函数来执行列表元素。订单列表大小可以使用 HistoryDealsTotal()，函数返回。使用HistoryOrdersTotal()。可以显示历史订单列表的大小。使用HistoryOrderGetTicket()，运行订单列表元素，订单列表元素—HistoryDealGetTicket()。

在使用 HistoryOrderSelect()， 后，MQL5程序中的历史可用订单列表就会重设并被发现的订单填满。如果成功执行 通过报价搜索订单同样适用于MQL5程序可行性订单中-通过HistoryDealSelect() 重设并再次成功填满。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   long  pos_id_array[];          // 用于存储仓位ID的数组

//--- 请求全部历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- 在数组中收集来自于挂单的仓位 ID
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
     {
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
```

© 2000-2025, MetaQuotes Ltd.

```
         continue;
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket, ORDER_TYPE)
      long pos_id=HistoryOrderGetInteger(ticket, ORDER_POSITION_ID);
      if(type<=ORDER_TYPE_SELL || pos_id==0)
         continue;

      int size=ArraySize(pos_id_array);
      if(ArrayResize(pos_id_array, size+1)==size+1)
         pos_id_array[size]=pos_id;
     }

//--- 根据数组中的仓位 ID 列表
   total=ArraySize(pos_id_array);
   for(int i=0; i<total; i++)
     {
      //--- 打印标题, 以及仓位的订单和交易列表
      long position_id=pos_id_array[i];
      Print("List of orders and deals for position with ID: ", position_id);
      HistorySelectByPositionProcess(position_id);
     }
   /*
   结果:
   List of orders and deals for position with ID: 1819629924
     [0] Order Sell Limit #1819629924
     [1] Order Buy #1819633194
     [0] Entry In Deal Sell #1794972472
     [1] Entry Out Deal Buy #1794975589
   List of orders and deals for position with ID: 1841753970
     [0] Order Sell Stop #1841753970
     [1] Order Buy #1842322160
     [0] Entry In Deal Sell #1817242142
     [1] Entry Out Deal Buy #1817765341
   */
  }
//+------------------------------------------------------------------+
//| 根据仓位ID选择订单和交易历史并                                   |
//| 在日志中打印该仓位的订单和交易列表                               |
//+------------------------------------------------------------------+
bool HistorySelectByPositionProcess(const long position_id)
  {
//--- 根据指定的仓位 ID 请求交易和订单的历史
   if(!HistorySelectByPosition(position_id))
     {
      PrintFormat("HistorySelectByPosition(%I64d) failed. Error %d", position_id, GetI
      return(false);
     }

//--- 打印仓位的订单列表
   int orders_total=HistoryOrdersTotal();
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0; i<orders_total; i++)
     {
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;
      ENUM_ORDER_TYPE order_type=(ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket, ORDE
      PrintFormat("  [%d] Order %s #%I64u", i, OrderTypeDescription(order_type), ticke
     }

//--- 在日志中打印仓位交易列表
   int deals_total =HistoryDealsTotal();
   for(int i=0; i<deals_total; i++)
     {
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;
      ENUM_DEAL_ENTRY deal_entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket, DEAL_E
      ENUM_DEAL_TYPE  deal_type= (ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, DEAL_TY
      if(deal_type!=DEAL_TYPE_BUY && deal_type!=DEAL_TYPE_SELL)
         continue;
      PrintFormat("  [%d] Entry %s Deal %s #%I64u", i, DealEntryDescription(deal_entry
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回仓位的交易类型描述                                           |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 只返回买入和卖出交易的描述,
      //--- 因为所有其它类型的交易并不适用于仓位
      case DEAL_TYPE_BUY    :  return("Buy");
      case DEAL_TYPE_SELL   :  return("Sell");
      default               :  return("Unknown deal type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回仓位变化的方法                                               |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN      :  return("In");
      case DEAL_ENTRY_OUT     :  return("Out");
      case DEAL_ENTRY_INOUT   :  return("InOut");
      case DEAL_ENTRY_OUT_BY  :  return("Out by");
      case DEAL_ENTRY_STATE   :  return("Status record");
      default                 :  return("Unknown deal entry: "+(string)entry);
     }
  }
```

另见

HistorySelect(), HistoryOrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrderSelect

从历史中选择订单，彻底调用适当函数。如果函数成功完成，返回真值，如果函数返回失败显示错误值。更多错误订单调用 GetLastError()。

```
bool  HistoryOrderSelect(
   ulong  ticket      // 订单号
   );
```

参量

*ticket*

[in] 订单号。

返回值

如果成功，返回真值，否则返回错误值。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

HistoryOrderSelect()在MQL5程序从历史记录中删除一系列订单，如果成功执行HistoryOrderSelect ()，可用调用和复制的单个字节，如果需要通过 HistorySelect()运行所有订单，最好使用 HistoryOrderGetTicket()。

示例：

```
#define   TICKET    1819621374   // 任何已知订单的编号, 例如来自账户历史中的订单
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 根据在 TICKET中指定的编号选择历史订单
   if(!HistoryOrderSelect(TICKET))
     {
      PrintFormat("HistoryOrderSelect(%I64u) failed. Error %d", TICKET, GetLastError()
      return;
     }

//--- 如果订单成功选中, 就取得它的数据并在日志中显示订单的描述
   ENUM_ORDER_TYPE   order_type  = (ENUM_ORDER_TYPE)HistoryOrderGetInteger(TICKET, ORI
   ENUM_ORDER_STATE  order_state = (ENUM_ORDER_STATE)HistoryOrderGetInteger(TICKET, OR
   ENUM_ORDER_REASON order_reason= (ENUM_ORDER_REASON)HistoryOrderGetInteger(TICKET, O
   long              order_time  = HistoryOrderGetInteger(TICKET, ORDER_TIME_SETUP_MSC
   string            order_symbol=HistoryOrderGetString(TICKET, ORDER_SYMBOL);
   double            order_vol_init=HistoryOrderGetDouble(TICKET, ORDER_VOLUME_INITIAL
   double            order_vol_curr=HistoryOrderGetDouble(TICKET, ORDER_VOLUME_CURRENT
   PrintFormat("%s Order %.2f/%s %s #%I64u %s by %s at %s",
               order_symbol, order_vol_init, (order_vol_curr>0 ? DoubleToString(order_
```

© 2000-2025, MetaQuotes Ltd.

```
                OrderTypeDescription(order_type), TICKET, OrderStateDescription(order_s
                OrderReasonDescription(order_reason), TimeMscToString(order_time));
   /*
   result for various specified tickets:
   EURUSD Order 0.50/0.50 Buy Limit #2812894647 Canceled by Client at 2024.09.04 19:02
   EURUSD Order 0.10/0 Sell #1753011743 Filled by Take Profit at 2023.06.12 17:04:20.3
   GBPUSD Order 0.10/0 Buy #1819621374 Filled by Client at 2023.07.24 06:16:25.746
   */
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| 返回订单状态的描述                                               |
//+------------------------------------------------------------------+
string OrderStateDescription(ENUM_ORDER_STATE state)
  {
   switch(state)
     {
      case ORDER_STATE_STARTED         :  return("Started");
      case ORDER_STATE_PLACED          :  return("Placed");
      case ORDER_STATE_CANCELED        :  return("Canceled");
      case ORDER_STATE_PARTIAL         :  return("Partial");
      case ORDER_STATE_FILLED          :  return("Filled");
      case ORDER_STATE_REJECTED        :  return("Rejected");
      case ORDER_STATE_EXPIRED         :  return("Expired");
      case ORDER_STATE_REQUEST_ADD     :  return("Request Add");
      case ORDER_STATE_REQUEST_MODIFY  :  return("Request Modify");
      case ORDER_STATE_REQUEST_CANCEL  :  return("Request Cancel");
      default                          :  return("Unknown state: "+(string)state);
     }
  }
//+------------------------------------------------------------------+
//| 返回下单原因的描述                                               |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
string OrderReasonDescription(const ENUM_ORDER_REASON reason)
  {
   switch(reason)
     {
      case ORDER_REASON_CLIENT    :  return("Client");
      case ORDER_REASON_MOBILE    :  return("Mobile");
      case ORDER_REASON_WEB       :  return("Web");
      case ORDER_REASON_EXPERT    :  return("Expert");
      case ORDER_REASON_SL        :  return("Stop Loss");
      case ORDER_REASON_TP        :  return("Take Profit");
      case ORDER_REASON_SO        :  return("Stop Out");
      default                     :  return("Unknown reason: "+(string)reason);
     }
  }
//+------------------------------------------------------------------+
//| 返回包含毫秒信息的时间                                           |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
```

另见

HistorySelect(), HistoryOrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrdersTotal

在历史中返回订单数量。优先调用HistoryOrdersTotal()，首先需要接收使用HistorySelect()或者HistorySelectByPosition()函数的交易和订单历史记录。

```
int  HistoryOrdersTotal();
```

返回值

整型值。

注释

不能把出现在客户端"工具箱"的"交易"标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求账户中的全部历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- 取得列表中的订单数量并在日志中显示
   int total=HistoryOrdersTotal();
   Print("Number of historical orders on the account: ", total);
   /*
  结果：
   Number of historical orders on the account: 496
   */
  }
```

另见

HistorySelect(), HistoryOrderSelect(), HistoryOrderGetTicket(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrderGetTicket

在历史中返回相关订单报价。优先调用HistoryOrderGetTicket()，首先需要接收使用 HistorySelect()或者 HistorySelectByPosition() 函数的交易和订单历史记录。

```
ulong  HistoryOrderGetTicket(
   int  index      // 订单列表中的数量
   );
```

参量

*index*

[in] 订单列表中的订单数。

返回值

无符长整型 值。如果函数失败，则返回0。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

示例：

```
void OnStart()
  {
   datetime from=0;
   datetime to=TimeCurrent();
//--- 要求全部历史记录
   HistorySelect(from,to);
//--- 返回订单属性值变量
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   datetime time_done;
   string   symbol;
   string   type;
   long     order_magic;
   long     positionID;
//--- 当前挂单数量
   uint     total=HistoryOrdersTotal();
//--- 循环检测通过订单
   for(uint i=0;i<total;i++)
     {
      //--- 通过其列表中的位置返回订单报价
      if((ticket=HistoryOrderGetTicket(i))>0)
        {
         //--- 返回订单属性
         open_price=       HistoryOrderGetDouble(ticket,ORDER_PRICE_OPEN);
         time_setup=       (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_SETUP);
```

© 2000-2025, MetaQuotes Ltd.

```
         time_done=        (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_DONE);
         symbol=           HistoryOrderGetString(ticket,ORDER_SYMBOL);
         order_magic=      HistoryOrderGetInteger(ticket,ORDER_MAGIC);
         positionID =      HistoryOrderGetInteger(ticket,ORDER_POSITION_ID);
         initial_volume=   HistoryOrderGetDouble(ticket,ORDER_VOLUME_INITIAL);
         type=GetOrderType(HistoryOrderGetInteger(ticket,ORDER_TYPE));
         //--- 准备并显示订单信息
         printf("#ticket %d %s %G %s at %G was set up at %s => done at %s, pos ID=%d",
                ticket,                   // 订单报价
                type,                     // 类型
                initial_volume,           // 已下的交易量
                symbol,                   // 交易品种
                open_price,               // 规定的开盘价
                TimeToString(time_setup),// 下订单时间
                TimeToString(time_done), // 订单执行或者删除时间
                positionID                // 一个包括订单交易的仓位ID
                );
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| 返回订单类型字符串名称                                           |
//+------------------------------------------------------------------+
string GetOrderType(long type)
  {
   string str_type="unknown operation";
   switch(type)
     {
      case (ORDER_TYPE_BUY):             return("buy");
      case (ORDER_TYPE_SELL):            return("sell");
      case (ORDER_TYPE_BUY_LIMIT):       return("buy limit");
      case (ORDER_TYPE_SELL_LIMIT):      return("sell limit");
      case (ORDER_TYPE_BUY_STOP):        return("buy stop");
      case (ORDER_TYPE_SELL_STOP):       return("sell stop");
      case (ORDER_TYPE_BUY_STOP_LIMIT): return("buy stop limit");
      case (ORDER_TYPE_SELL_STOP_LIMIT):return("sell stop limit");
     }
   return(str_type);
  }
```

另见

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrderGetDouble

返回要求的订单属性，订单属性一定是双精度型。有2个变量属性可供使用。

1. 立即返回属性值。

```
double  HistoryOrderGetDouble(
   ulong                        ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_DOUBLE   property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryOrderGetDouble(
   ulong                        ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_DOUBLE   property_id,       // 属性标识符
   double&                      double_var         // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 订单号。

*property_id*

[in] 订单属性标识符。值可以是 ENUM_ORDER_PROPERTY_DOUBLE 列举值中的一个。

*double_var*

[out] 接收需要的属性值双精度类型变量。

返回值

双精度 类型值。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求交易和订单历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在账户的全部历史订单中做循环
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- 根据循环索引取得列表中的订单编号
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得订单类型并显示用于选中订单实数型属性列表的标题
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket,
      PrintFormat("Double properties of an history order %s #%I64u:", type, ticket);

      //--- 在标题下方打印选中订单的全部实数型属性
      HistoryOrderPropertiesDoublePrint(ticket, 16);
     }
   /*
  结果:
   Double properties of an history order Sell #2810847541:
   Volume initial: 0.50
   Volume current: 0.00
   Price open:     1.10491
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10491
   StopLimit:      0.00000
   Double properties of an history order Buy Limit #2811003507:
   Volume initial: 1.00
   Volume current: 1.00
   Price open:     1.10547
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10591
   StopLimit:      0.00000
   */
  }
//+------------------------------------------------------------------+
//| 显示实数型属性                                                   |
//| 适用于日志和选中的历史订单                                       |
//+------------------------------------------------------------------+
void HistoryOrderPropertiesDoublePrint(const long ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;

//--- 取得订单的交易品种和交易品种的小数位数
   string symbol = HistoryOrderGetString(ticket, ORDER_SYMBOL);
```

© 2000-2025, MetaQuotes Ltd.

```
   int    digits = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0, 则宽度为标题文字行的大小+1
   header="Volume initial:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中以指定宽度显示下订单时的初始交易量
   if(!HistoryOrderGetDouble(ticket, ORDER_VOLUME_INITIAL, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示未执行的订单交易量
   header="Volume current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_VOLUME_CURRENT, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示订单中指定的价格
   header="Price open:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_OPEN, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示止损水平
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示止盈水平
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示订单交易品种的当前价格
   header="Price current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_CURRENT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示限价止损订单的订单限价
   header="StopLimit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_STOPLIMIT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
```

另见

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrderGetInteger

返回订单的要求属性，订单属性一定是日期时间，整型，有2种变量函数可供使用。

1. 立即返回属性值。

```
long  HistoryOrderGetInteger(
   ulong                          ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_INTEGER    property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryOrderGetInteger(
   ulong                          ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_INTEGER    property_id,       // 属性标识符
   long&                          long_var           // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 订单号。

*property_id*

[in] 订单属性标识符，值可以是 ENUM_ORDER_PROPERTY_INTEGER 列举值中的一个。

*long_var*

[out] 接收要求属性值的长型变量。

返回值

长整型 值。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 交易函数                                                         |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- 从周交易历史记录接收最后订单的标签
   ulong last_order=GetLastOrderTicket();
   if(HistoryOrderSelect(last_order))
     {
      //--- 从1970.01.01起以毫秒为单位下单的时间
      long time_setup_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_SETUP_MSC);
      PrintFormat("Order #%d ORDER_TIME_SETUP_MSC=%i64 => %s",
                  last_order,time_setup_msc,TimeToString(time_setup_msc/1000));
      //--- 从1970.01.01起以毫秒为单位执行/取消订单的时间
      long  time_done_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_DONE_MSC);
      PrintFormat("Order #%d ORDER_TIME_DONE_MSC=%i64 => %s",
                  last_order,time_done_msc,TimeToString(time_done_msc/1000));
     }
   else // 通知失败
      PrintFormat("HistoryOrderSelect() failed for #%d. Eror code=%d",
                  last_order,GetLastError());

//---
  }
//+------------------------------------------------------------------+
//| 返回历史记录中最后订单的标签或-1                                 |
//+------------------------------------------------------------------+
ulong GetLastOrderTicket()
  {
//--- 请求最近7天的历史记录
   if(!GetTradeHistory(7))
     {
      //--- 通知不成功的调用和返回-1
      Print(__FUNCTION__," HistorySelect() returned false");
      return -1;
     }
//---
   ulong first_order,last_order,orders=HistoryOrdersTotal();
//--- 如果有，从事订单工作
   if(orders>0)
     {
      Print("Orders = ",orders);
      first_order=HistoryOrderGetTicket(0);
      PrintFormat("first_order = %d",first_order);
      if(orders>1)
        {
         last_order=HistoryOrderGetTicket((int)orders-1);
         PrintFormat("last_order = %d",last_order);
         return last_order;
        }
      return first_order;
     }
//--- 未发现订单，返回 -1
   return -1;
  }
//+------------------------------------------------------------------+
//| 请求最近几日的历史记录，如果失败，则返回false                    |
//+------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- 设置一个星期的周期请求交易历史记录
```

© 2000-2025, MetaQuotes Ltd.

```
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- 发出请求检察结果
   if(!HistorySelect(from,to))
     {
      Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
      return false;
     }
//--- 成功接收历史记录
   return true;
  }
```

另 见

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), 订 单 属 性

© 2000-2025, MetaQuotes Ltd.

# HistoryOrderGetString

返回订单要求属性，订单属性一定是字符串类型，有2个变量函数可供使用。

1. 立即返回属性值。

```
string  HistoryOrderGetString(
   ulong                       ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_STRING  property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryOrderGetString(
   ulong                       ticket_number,     // 订单号
   ENUM_ORDER_PROPERTY_STRING  property_id,       // 属性标识符
   string&                     string_var         // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 订单号。

*property_id*

[in] 订单属性标识符。值可以是 ENUM_ORDER_PROPERTY_STRING 列举值中的一个。

*string_var*

[out] 字符串型变量。

返回值

字符串 型值。

注释

不能把出现在客户端"工具箱"的"交易" 标签中的当前挂单和交易历史订单弄乱。取消的或者导致的交易的订单列表可以在"工具箱"客户端中的"历史记录"标签浏览。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求交易和订单历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在账户的全部历史订单中做循环
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- 根据循环索引取得列表中的订单编号
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得订单类型并显示用于选中订单字符串型属性列表的标题
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket,
      PrintFormat("String properties of an history order %s #%I64u:", type, ticket);

      //--- 在标题下方打印选中订单的所有字符串型属性
      HistoryOrderPropertiesStringPrint(ticket, 16);
     }
   /*
  结果:
   String properties of an history order Buy #2646074112:
   Comment:         [tp 1.09137]
   Symbol:          EURUSD
   External ID:
   String properties of an history order Buy #2646131906:
   Comment:
   Symbol:          EURUSD
   External ID:
   */
  }
//+------------------------------------------------------------------+
//| 显示字符串型属性                                                 |
//| 适用于日志和选中的历史订单                                       |
//+------------------------------------------------------------------+
void HistoryOrderPropertiesStringPrint(const long ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value="";

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0, 则宽度为标题文字行的大小+1
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中使用指定的标题宽度显示注释
   if(!HistoryOrderGetString(ticket, ORDER_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 在日志中显示所下订单的交易品种
```

© 2000-2025, MetaQuotes Ltd.

```
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetString(ticket, ORDER_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 在日志中显示在外部系统中的订单 ID
   header="External ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetString(ticket, ORDER_EXTERNAL_ID, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
  }
//+------------------------------------------------------------------+
//| 返回订单类型的描述                                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
```

另见

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), 订单属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealSelect

通过调用恰当函数选择历史交易。如果函数成功完成，返回真值，如果函数失败返回错误值。有关错误更多细节，调用 GetLastError()。

```
bool  HistoryDealSelect(
   ulong  ticket      // 交易订单号
   );
```

参量

*ticket*

[in]交易订单号。

返回值

如果成功返回真值，否则返回错误值。

注释

不要弄混 订单，交易 和 仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

HistoryDealSelect()在MQL5程序从历史记录中删除一系列订单，如果成功执行HistoryDealSelect()，可用调用和复制的单个交易，如果需要通过 HistorySelect() 运行所有订单，最好使用 HistoryDealGetTicket()。

示例：

```
#define   TICKET    2620919264   // 任何已知交易的编号，例如来自于终端账户历史中的交易

long      ExtTicket=TICKET;      // 把脚本测试中宏替换的指定编号设置到变量中，
                                 // 该宏替换也可能用于在 EA 中的 OnTradeTransaction() 处理
//+------------------------------------------------------------------+
//| Expert TradeTransaction handler                                  |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
  {
   //--- 如果事务是在历史中添加一个交易
   if(trans.type==TRADE_TRANSACTION_DEAL_ADD)
     {
      //--- 根据编号选择交易，取得它的数据并在日志中显示交易的描述
      HistoryDealSelectProcess(trans.deal);
     }
  }

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 根据编号选择交易，取得它的数据并在日志中显示交易的描述
   HistoryDealSelectProcess(ExtTicket);
   /*
  结果：
   (Position ID #2645974677) EURUSD Deal Out 0.10 Buy #2620919264 by order #264602896
   */
  }
//+------------------------------------------------------------------+
//| 根据编号选择交易并在日志中打印交易数据                                   |
//+------------------------------------------------------------------+
void HistoryDealSelectProcess(const ulong deal_ticket)
  {
//--- 根据在 deal_ticket 中指定的编号选择历史交易
   ResetLastError();
   if(!HistoryDealSelect(deal_ticket))
     {
      PrintFormat("HistoryDealSelect(%I64u) failed. Error %d", deal_ticket, GetLastErr
      return;
     }

//--- 如果成功选择了交易，就取得它的数据并在日志中显示交易的描述
   ENUM_DEAL_TYPE    deal_type   = (ENUM_DEAL_TYPE)HistoryDealGetInteger(ExtTicket, DEA
   ENUM_DEAL_ENTRY   deal_entry  = (ENUM_DEAL_ENTRY)HistoryDealGetInteger(ExtTicket, DE
   ENUM_DEAL_REASON  deal_reason= (ENUM_DEAL_REASON)HistoryDealGetInteger(ExtTicket, D
   long              deal_time   = HistoryDealGetInteger(ExtTicket, DEAL_TIME_MSC);
   long              deal_order  = HistoryDealGetInteger(ExtTicket, DEAL_ORDER);
   long              deal_pos_id= HistoryDealGetInteger(ExtTicket, DEAL_POSITION_ID);
   string            deal_symbol= HistoryDealGetString(ExtTicket, DEAL_SYMBOL);
   double            deal_volume= HistoryDealGetDouble(ExtTicket, DEAL_VOLUME);
   double            deal_price  = HistoryDealGetDouble(ExtTicket, DEAL_PRICE);
   int               digits      = (int)SymbolInfoInteger(deal_symbol, SYMBOL_DIGITS);

   PrintFormat("(Position ID #%I64d) %s Deal %s %.2f %s #%I64u by order #%I64d at %.*
               deal_pos_id, deal_symbol, DealEntryDescription(deal_entry), deal_volume
               DealTypeDescription(deal_type), ExtTicket, deal_order, digits, deal_pri
               TimeMscToString(deal_time));
  }
//+------------------------------------------------------------------+
//| 返回交易类型的描述                                                  |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                    :  return("Buy");
      case DEAL_TYPE_SELL                   :  return("Sell");
      case DEAL_TYPE_BALANCE                :  return("Balance");
      case DEAL_TYPE_CREDIT                 :  return("Credit");
```

© 2000-2025, MetaQuotes Ltd.

```
      case DEAL_TYPE_CHARGE                   :  return("Additional charge");
      case DEAL_TYPE_CORRECTION               :  return("Correction");
      case DEAL_TYPE_BONUS                    :  return("Bonus");
      case DEAL_TYPE_COMMISSION               :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY         :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY       :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY   :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY:  return("Monthly agent commission");
      case DEAL_TYPE_INTEREST                 :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED             :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED            :  return("Canceled sell deal");
      case DEAL_DIVIDEND                      :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED              :  return("Franked (non-taxable) dividend
      case DEAL_TAX                           :  return("Tax charges");
      default                                 :  return("Unknown deal type: "+(string)t
     }
  }
//+------------------------------------------------------------------+
//| 返回仓位变化的方法                                                 |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN      :  return("In");
      case DEAL_ENTRY_OUT     :  return("Out");
      case DEAL_ENTRY_INOUT   :  return("Reverce");
      case DEAL_ENTRY_OUT_BY  :  return("Out by");
      case DEAL_ENTRY_STATE   :  return("Status record");
      default                 :  return("Unknown deal entry: "+(string)entry);
     }
  }
//+------------------------------------------------------------------+
//| 返回包含毫秒信息的时间                                             |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
```

另见

HistorySelect(), HistoryDealGetTicket(), 交易属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealsTotal

返回历史交易数量，先前调用HistoryDealsTotal()，首先有必要接收交易历史和使用HistorySelect()或者HistorySelectByPosition()函数的订单。

```
int  HistoryDealsTotal();
```

返回值

整型值。

注释

不要弄混订单，交易和仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求账户中的全部历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- 取得列表中的交易数量并在日志中显示
   int total=HistoryDealsTotal();
   Print("Number of historical deals on the account: ", total);
   /*
  结果：
   Number of historical deals on the account: 339
   */
  }
```

另见

HistorySelect()，HistoryDealGetTicket()，交易属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealGetTicket

函数选择进一步交易过程并返回历史交易票据。优先调用HistoryDealGetTicket()，首先有必要接收使用 HistorySelect() 或者 HistorySelectByPosition() 函数的交易和订单历史记录 。

```
ulong  HistoryDealGetTicket(
   int  index      // 订单号交易
   );
```

参量

*index*

[in] 交易列表中的交易数量。

返回值

无符长整型 值。如果函数失败，则返回0。

注释

不要弄混 订单, 交易 和 仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

示例：

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
   ulong deal_ticket;             // 交易订单号
   ulong order_ticket;            // 执行交易的订单的订单号
   datetime transaction_time;     // 交易执行时间
   long deal_type ;               // 交易操作类型
   long position_ID;              // 仓位ID
   string deal_description;       // 操作描述
   double volume;                 // 操作交易量
   string symbol;                 // 交易的交易品种
//--- 设置交易历史记录请求的起止时间
   datetime from_date=0;          // 从最开始
   datetime to_date=TimeCurrent();// 到当前时间
//--- 特定周期中请求交易历史记录
   HistorySelect(from_date,to_date);
//--- 交易列表中的全部数量
   int deals=HistoryDealsTotal();
//--- 现在处理每一个交易
   for(int i=0;i<deals;i++)
     {
      deal_ticket=               HistoryDealGetTicket(i);
      volume=                    HistoryDealGetDouble(deal_ticket,DEAL_VOLUME);
      transaction_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
      order_ticket=              HistoryDealGetInteger(deal_ticket,DEAL_ORDER);
      deal_type=                 HistoryDealGetInteger(deal_ticket,DEAL_TYPE);
      symbol=                    HistoryDealGetString(deal_ticket,DEAL_SYMBOL);
      position_ID=               HistoryDealGetInteger(deal_ticket,DEAL_POSITION_ID);
      deal_description=          GetDealDescription(deal_type,volume,symbol,order_tic
      //--- 为交易号执行格式化
      string print_index=StringFormat("% 3d",i);
      //--- 显示交易信息
      Print(print_index+": deal #",deal_ticket," at ",transaction_time,deal_descriptio
     }
  }
//+------------------------------------------------------------------+
//| 返回操作的字符串描述                                             |
//+------------------------------------------------------------------+
string GetDealDescription(long deal_type,double volume,string symbol,long ticket,long
  {
   string descr;
//---
   switch(deal_type)
     {
      case DEAL_TYPE_BALANCE:                  return ("balance");
      case DEAL_TYPE_CREDIT:                   return ("credit");
      case DEAL_TYPE_CHARGE:                   return ("charge");
      case DEAL_TYPE_CORRECTION:               return ("correction");
      case DEAL_TYPE_BUY:                      descr="buy"; break;
      case DEAL_TYPE_SELL:                     descr="sell"; break;
      case DEAL_TYPE_BONUS:                    return ("bonus");
      case DEAL_TYPE_COMMISSION:               return ("additional commission");
      case DEAL_TYPE_COMMISSION_DAILY:         return ("daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY:       return ("monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY:   return ("daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return ("monthly agent commission");
      case DEAL_TYPE_INTEREST:                 return ("interest rate");
      case DEAL_TYPE_BUY_CANCELED:             descr="cancelled buy deal"; break;
      case DEAL_TYPE_SELL_CANCELED:            descr="cancelled sell deal"; break;
     }
   descr=StringFormat("%s %G %s (order #%d, position ID %d)",
                      descr,  // 当前描述。
```

© 2000-2025, MetaQuotes Ltd.

```
                        volume, // 交易的交易量。
                        symbol, // 交易的交易品种。
                        ticket, // 引起交易的订单的订单号。
                        pos_ID  // 仓位交易执行的ID号。
                        );
   return(descr);
//---
  }
```

另见

HistorySelect(), HistoryDealsTotal(), HistoryDealSelect(), 交易属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealGetDouble

返回要求的交易属性，交易属性一定是双精度型，有2中变量函数可供使用。

1. 立即返回属性值。

```
double  HistoryDealGetDouble(
   ulong                      ticket_number,     // 订单号
   ENUM_DEAL_PROPERTY_DOUBLE  property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryDealGetDouble(
   ulong                      ticket_number,     // 订单号
   ENUM_DEAL_PROPERTY_DOUBLE  property_id,       // 属性标识符
   double&                    double_var         // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 交易订单号。

*property_id*

[in] 交易属性标识符。值可以是 ENUM_DEAL_PROPERTY_DOUBLE enumeration列举值中的一个。

*double_var*

[out] 接收要求属性值的双精度类型变量。

返回值

双精度 型值。

注释

不要弄混 订单, 交易 和 仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求交易和订单历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 根据账户历史中的交易列表做循环
   int total=HistoryDealsTotal();
   for(int i=0; i<total; i++)
     {
      //--- 取得下一个交易的编号（自动选择交易以取得它的属性）
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得交易的类型和方向并显示选中交易实数属性列表的标题
      string type=DealTypeDescription((ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, DE
      string entry=DealEntryDescription((ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket,
      PrintFormat("Double properties of an deal %s entry %s #%I64u:", type, entry, tic

      //--- 在标题下方打印选中交易的全部实数类型属性
      HistoryDealPropertiesDoublePrint(ticket, 12);
     }
   /*
   real:
   Double properties of an deal Buy entry In #2785070622:
   Volume:      0.50
   Price:       1.10480
   Commission: 0.00
   Swap:        0.00
   Profit:      0.00 USD
   Fee:         0.00
   StopLoss:    0.00000
   TakeProfit: 0.00000
   Double properties of an deal Sell entry Out #2785071538:
   Volume:      0.50
   Price:       1.10491
   Commission: 0.00
   Swap:        0.00
   Profit:      5.50 USD
   Fee:         0.00
   StopLoss:    0.00000
   TakeProfit: 0.00000
   */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选中交易的实数类型属性                                |
//+------------------------------------------------------------------+
void HistoryDealPropertiesDoublePrint(const ulong ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 取得交易的交易品种，利润货币和交易品种的小数点位数
   string symbol   = HistoryDealGetString(ticket, DEAL_SYMBOL);
   string currency= SymbolInfoString(symbol, SYMBOL_CURRENCY_PROFIT);
   int    digits   = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0，则宽度为标题文字行的大小+1
   header="Volume:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中使用指定的标题宽度显示交易量
   if(!HistoryDealGetDouble(ticket, DEAL_VOLUME, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示价格
   header="Price:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_PRICE, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示交易佣金
   header="Commission:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_COMMISSION, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示平仓时的累积库存费
   header="Swap:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_SWAP, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- 在日志中显示交易的财务结果
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_PROFIT, value))
      return;
   PrintFormat("%-*s%-.2f %s", w, header, value, currency);

//--- 在日志中显示交易费用
   header="Fee:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_FEE, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 在日志中显示止损水平
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- 在日志中显示止盈水平
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);
  }
//+------------------------------------------------------------------+
//| 返回交易类型的描述                                               |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                    :  return("Buy");
      case DEAL_TYPE_SELL                   :  return("Sell");
      case DEAL_TYPE_BALANCE                :  return("Balance");
      case DEAL_TYPE_CREDIT                 :  return("Credit");
      case DEAL_TYPE_CHARGE                 :  return("Additional charge");
      case DEAL_TYPE_CORRECTION             :  return("Correction");
      case DEAL_TYPE_BONUS                  :  return("Bonus");
      case DEAL_TYPE_COMMISSION             :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY       :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY     :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return("Monthly agent commission");
      case DEAL_TYPE_INTEREST               :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED           :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED          :  return("Canceled sell deal");
      case DEAL_DIVIDEND                    :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED            :  return("Franked (non-taxable) dividend
      case DEAL_TAX                         :  return("Tax charges");
      default                               :  return("Unknown deal type: "+(string)t
     }
  }
//+------------------------------------------------------------------+
//| 返回仓位变化的方法                                               |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
```

© 2000-2025, MetaQuotes Ltd.

```
     {
      case DEAL_ENTRY_IN      :  return("In");
      case DEAL_ENTRY_OUT     :  return("Out");
      case DEAL_ENTRY_INOUT   :  return("Reverce");
      case DEAL_ENTRY_OUT_BY  :  return("Out by");
      case DEAL_ENTRY_STATE   :  return("Status record");
      default                 :  return("Unknown deal entry: "+(string)entry);
     }
  }
```

另见

HistorySelect(), HistoryDealsTotal(), HistoryDealGetTicket(), 交易属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealGetInteger

返回要求交易的属性，交易属性一定是日期时间，整型。有2个变量函数可供使用。

1. 立即返回属性值。

```
long  HistoryDealGetInteger(
   ulong                      ticket_number,     // 订单号
   ENUM_DEAL_PROPERTY_INTEGER  property_id       // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryDealGetInteger(
   ulong                      ticket_number,     // 订单号
   ENUM_DEAL_PROPERTY_INTEGER  property_id,      // 属性标识符
   long&                      long_var           // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 交易订单号。

*property_id*

[in] 交易属性标识符。值可以是 ENUM_DEAL_PROPERTY_INTEGER 列举值中的一个。

*long_var*

[out] 接收要求属性值的长型变量。

返回值

长整型 值。

注释

不要弄混 订单，交易 和 仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

例如：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 交易函数                                                         |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- 从周交易历史记录接收最后订单的标签
   ulong last_deal=GetLastDealTicket();
   if(HistoryDealSelect(last_deal))
     {
      //--- 从1970.01.01起以毫秒为单位的交易执行的时间
      long deal_time_msc=HistoryDealGetInteger(last_deal,DEAL_TIME_MSC);
      PrintFormat("Deal #%d DEAL_TIME_MSC=%i64 => %s",
                  last_deal,deal_time_msc,TimeToString(deal_time_msc/1000));
     }
   else
      PrintFormat("HistoryDealSelect() failed for #%d. Eror code=%d",
                  last_deal,GetLastError());
//---
  }
//+------------------------------------------------------------------+
//| 返回历史记录中最后交易的标签或-1                                 |
//+------------------------------------------------------------------+
ulong GetLastDealTicket()
  {
//--- 请求最近7天的历史记录
   if(!GetTradeHistory(7))
     {
      //--- 通知不成功的调用和返回-1
      Print(__FUNCTION__," HistorySelect() returned false");
      return -1;
     }
//---
   ulong first_deal,last_deal,deals=HistoryDealsTotal();
//--- 如果有，从事订单工作
   if(deals>0)
     {
      Print("Deals = ",deals);
      first_deal=HistoryDealGetTicket(0);
      PrintFormat("first_deal = %d",first_deal);
      if(deals>1)
        {
         last_deal=HistoryDealGetTicket((int)deals-1);
         PrintFormat("last_deal = %d",last_deal);
         return last_deal;
        }
      return first_deal;
     }
//--- 未发现成交，返回 -1
   return -1;
  }
//+------------------------------------------------------------------+
//| 请求最近几日的历史记录，如果失败，则返回false                    |
//+------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- 设置一个星期的周期请求交易历史记录
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- 发出请求检察结果
   if(!HistorySelect(from,to))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
       return false;
      }
//--- 成功接收历史记录
   return true;
  }
```

另见

HistorySelect(), HistoryDealsTotal(), HistoryDealGetTicket(), 交易属性

© 2000-2025, MetaQuotes Ltd.

# HistoryDealGetString

返回要求的交易属性，交易属性一定是字符串型，有2个变量函数可供使用。

1. 立即返回属性值。

```
string  HistoryDealGetString(
   ulong                      ticket_number,     // 订单号
   ENUM_DEAL_PROPERTY_STRING  property_id        // 属性标识符
   );
```

2. 依据函数成功与否返回真值或者错误值。如果成功，目标变量的属性值通过引用传递到最后参量。

```
bool  HistoryDealGetString(
   ulong                      ticket_number,     //订单号
   ENUM_DEAL_PROPERTY_STRING  property_id,       // 属性标识符
   string&                    string_var         // 这里接受属性值
   );
```

参量

*ticket_number*

[in] 交易订单号。

*property_id*

[in] 交易属性标识符。值可以是 ENUM_DEAL_PROPERTY_STRING 列举值中的一个。

*string_var*

[out] 接收要求属性值的字符串型变量。

返回值

字符串型 值。

注释

不要弄混 订单，交易 和 仓位。每笔交易都按照订单执行，每个仓位都是一个或多个交易的值。

示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 请求交易和订单历史
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 根据账户历史中的交易列表做循环
   int total=HistoryDealsTotal();
   for(int i=0; i<total; i++)
     {
      //--- 取得下一个交易的编号（自动选择交易以取得它的属性）
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;

      //--- 取得交易的类型和方向并显示选中交易实数属性列表的标题
      string type=DealTypeDescription((ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, DE
      string entry=DealEntryDescription((ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket,
      PrintFormat("String properties of an deal %s entry %s #%I64u:", type, entry, tic

      //--- 在标题下方打印选中交易的全部实数类型属性
      HistoryDealPropertiesStringPrint(ticket, 13);
     }
   /*
  结果：
   String properties of an deal Buy entry In #2785021084:
   Symbol:       EURUSD
   Comment:      Test PositionGetString
   Extarnal ID:
   String properties of an deal Buy entry Out #2497993663:
   Symbol:       EURUSD
   Comment:      [tp 1.08639]
   Extarnal ID:
   */
  }
//+------------------------------------------------------------------+
//| 在日志中显示选中交易的字符串型属性                                 |
//+------------------------------------------------------------------+
void HistoryDealPropertiesStringPrint(const ulong ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value ="";

//--- 定义标题文字和标题栏位的宽度
//--- 如果传给函数的标题宽度为0，则宽度为标题文字行的大小+1
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- 取得并在日志中以指定标题宽度显示交易的交易品种
   if(!HistoryDealGetString(ticket, DEAL_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 在日志中显示交易的注释
```

© 2000-2025, MetaQuotes Ltd.

```
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetString(ticket, DEAL_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- 显示外部交易系统中的交易 ID
   header="Extarnal ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetString(ticket, DEAL_EXTERNAL_ID, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
  }
//+------------------------------------------------------------------+
//| 返回交易类型的描述                                               |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                     :  return("Buy");
      case DEAL_TYPE_SELL                    :  return("Sell");
      case DEAL_TYPE_BALANCE                 :  return("Balance");
      case DEAL_TYPE_CREDIT                  :  return("Credit");
      case DEAL_TYPE_CHARGE                  :  return("Additional charge");
      case DEAL_TYPE_CORRECTION              :  return("Correction");
      case DEAL_TYPE_BONUS                   :  return("Bonus");
      case DEAL_TYPE_COMMISSION              :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY        :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY      :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY  :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY:  return("Monthly agent commission");
      case DEAL_TYPE_INTEREST                :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED            :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED           :  return("Canceled sell deal");
      case DEAL_DIVIDEND                     :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED             :  return("Franked (non-taxable) dividenc
      case DEAL_TAX                          :  return("Tax charges");
      default                                :  return("Unknown deal type: "+(string)t
     }
  }
//+------------------------------------------------------------------+
//| 返回仓位变化的方法                                               |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN      :  return("In");
```

© 2000-2025, MetaQuotes Ltd.

```
      case DEAL_ENTRY_OUT     :  return("Out");
      case DEAL_ENTRY_INOUT   :  return("Reverce");
      case DEAL_ENTRY_OUT_BY  :  return("Out by");
      case DEAL_ENTRY_STATE   :  return("Status record");
      default                 :  return("Unknown deal entry: "+(string)entry);
     }
  }
```

另见

HistorySelect(), HistoryDealsTotal(), HistoryDealGetTicket(), 交易属性

© 2000-2025, MetaQuotes Ltd.

# 交易信号

这是意于管理交易信号的函数组。该函数允许：

- 获得交易信号的信息，可用于复制，
- 获得并设置信号复制设置，
- 使用MQL5语言函数订阅和退订信号复制。

| 函数 | 功能 |
| --- | --- |
| SignalBaseGetDouble | 返回选定信号double类型属性的值 |
| SignalBaseGetInteger | 返回选定信号integer类型属性的值 |
| SignalBaseGetString | 返回选定信号string类型属性的值 |
| SignalBaseSelect | 选择来自信号组的信号，在程序端可以得到，用于进一步的工作 |
| SignalBaseTotal | 返回信号总额，在程序端可得 |
| SignalInfoGetDouble | 返回信号复制设置的double类型属性的值 |
| SignalInfoGetInteger | 返回信号复制设置的integer类型属性的值 |
| SignalInfoGetString | 返回信号复制设置的string类型属性的值 |
| SignalInfoSetDouble | 设置信号复制设置的double类型属性的值 |
| SignalInfoSetInteger | 设置信号复制设置的integer类型属性的值 |
| SignalSubscribe | 订阅交易信号 |
| SignalUnsubscribe | 取消订阅 |

© 2000-2025, MetaQuotes Ltd.

# SignalBaseGetDouble

返回选定信号的 double 类型属性的值。

```
double  SignalBaseGetDouble(
   ENUM_SIGNAL_BASE_DOUBLE     property_id,     // 属性标识符
   );
```

参数

*property_id*

[in] 信号属性标识符。该值可以是 ENUM_SIGNAL_BASE_DOUBLE 枚举值之一。

返回值

选定信号 double 类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalBaseGetInteger

返回选定信号的integer类型属性的值。

```
long  SignalBaseGetInteger(
   ENUM_SIGNAL_BASE_INTEGER     property_id,     // 属性标识符
   );
```

参数

*property_id*

[in] 信号属性标识符。该值可以是ENUM_SIGNAL_BASE_INTEGER枚举值之一。

返回值

选定信号integer类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalBaseGetString

返回选定信号的string类型属性的值。

```
string  SignalBaseGetString(
   ENUM_SIGNAL_BASE_STRING     property_id,     // 属性标识符
   );
```

参数

*property_id*

[in] 信号属性标识符。该值可以是 ENUM_SIGNAL_BASE_STRING 枚举值之一。

返回值

选定信号string类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalBaseSelect

选择来自信号组的信号，在程序端可以得到，用于进一步的工作。

```
bool  SignalBaseSelect(
   int      index     // 信号指数
   );
```

参数

*index*

[in] 基于交易信号的信号指数。

返回值

如果成功返回true，否则返回false。要阅读更多错误信息调用GetLastError()。

示例：

```
void OnStart()
  {
//--- 获得程序端信号的总数
   int total=SignalBaseTotal();
//--- 处理所有信号
   for(int i=0;i<total;i++)
     {
      //--- 选择指数信号
      if(SignalBaseSelect(i))
        {
         //--- 获得信号属性
         long   id    =SignalBaseGetInteger(SIGNAL_BASE_ID);           // 信号id
         long   pips  =SignalBaseGetInteger(SIGNAL_BASE_PIPS);         // 利润点
         long   subscr=SignalBaseGetInteger(SIGNAL_BASE_SUBSCRIBERS); // 订阅人数量
         string name  =SignalBaseGetString(SIGNAL_BASE_NAME);          // 信号名称
         double price =SignalBaseGetDouble(SIGNAL_BASE_PRICE);         // 信号价格
         string curr  =SignalBaseGetString(SIGNAL_BASE_CURRENCY);      // signal currer
         //--- 打印所有订阅人的获利免费信号
         if(price==0.0 && pips>0 && subscr>0)
            PrintFormat("id=%d, name=\"%s\", currency=%s, pips=%d, subscribers=%d",id,
        }
      else PrintFormat("Error in call of SignalBaseSelect. Error code=%d",GetLastErro
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# SignalBaseTotal

返回信号总额，在程序端可得。

```
int  SignalBaseTotal();
```

返回值

信号总额，在程序端可得。

© 2000-2025, MetaQuotes Ltd.

# SignalInfoGetDouble

返回信号复制设置的 double 类型属性的值。

```
double  SignalInfoGetDouble(
   ENUM_SIGNAL_INFO_DOUBLE     property_id,      // 属性标识符
   );
```

参数

*property_id*

[in] 信号复制设置属性标识符。该值可以是 ENUM_SIGNAL_INFO_DOUBLE 枚举值之一。

返回值

信号复制设置的 double 类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalInfoGetInteger

返回信号复制设置的integer类型属性的值。

```
long  SignalInfoGetInteger(
   ENUM_SIGNAL_INFO_INTEGER     property_id,     // 属性标识符
   );
```

参数

*property_id*

[in] 信号复制设置属性标识符。该值可以是 ENUM_SIGNAL_INFO_INTEGER 枚举值之一。

返回值

信号复制设置的integer 类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalInfoGetString

返回信号复制设置的string类型属性的值。

```
string  SignalInfoGetString(
   ENUM_SIGNAL_INFO_STRING     property_id,     // 属性标识符
   );
```

参数

*property_id*

[in] 信号复制设置属性标识符。该值可以是 ENUM_SIGNAL_INFO_STRING 枚举值之一。

返回值

信号复制设置的 string 类型属性的值。

© 2000-2025, MetaQuotes Ltd.

# SignalInfoSetDouble

设置信号复制设置的 double 类型属性的值。

```
bool  SignalInfoSetDouble(
   ENUM_SIGNAL_INFO_DOUBLE       property_id,      // 属性标识符
   double                        value             // 新值
   );
```

参数

*property_id*

[in] 信号复制设置属性标识符。该值可以是 ENUM_SIGNAL_INFO_DOUBLE 枚举值之一。

*value*

[in] 信号复制设置属性的值。

返回值

如果属性改变返回true，否则返回 false。要阅读更多错误信息调用GetLastError() 。

© 2000-2025, MetaQuotes Ltd.

# SignalInfoSetInteger

设置信号复制设置的integer类型属性的值。

```
bool  SignalInfoSetInteger(
   ENUM_SIGNAL_INFO_INTEGER     property_id,     // 属性标识符
   long                         value            // 新值
   );
```

参数

*property_id*

[in] 信号复制设置属性标识符。该值可以是 ENUM_SIGNAL_INFO_INTEGER 枚举值之一。

*value*

[in] 信号复制设置属性的值。

返回值

如果属性改变返回true，否则返回false。要阅读更多错误信息调用GetLastError()。

© 2000-2025, MetaQuotes Ltd.

# SignalSubscribe

订阅交易信号。

```
bool  SignalSubscribe(
   long     signal_id     // 信号 id
   );
```

参数

*signal_id*

[in] 信号标识符。

返回值

如果订阅成功返回true，否则返回false。要阅读更多错误信息调用GetLastError()。

© 2000-2025, MetaQuotes Ltd.

# SignalUnsubscribe

取消订阅。

```
bool  SignalUnsubscribe();
```

返回值

如果订阅成功取消返回true，否则返回false。要阅读更多错误信息调用GetLastError()。

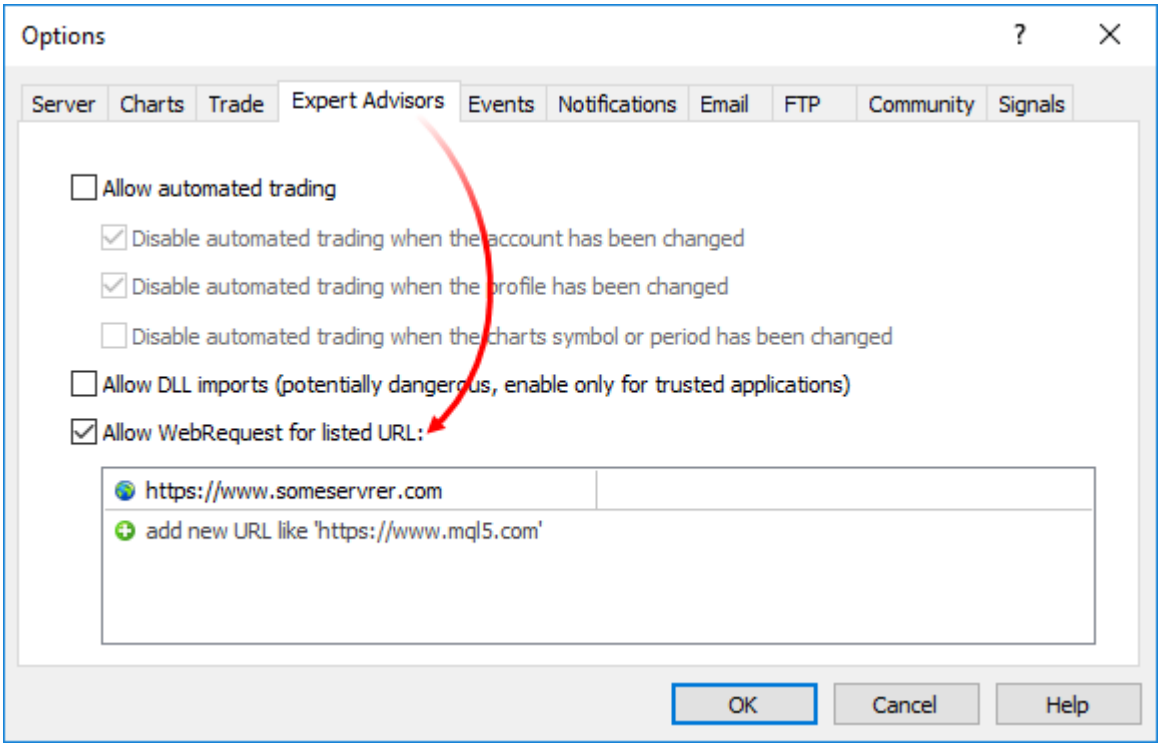
© 2000-2025, MetaQuotes Ltd.

# 网络函数

MQL5程序可以与远程服务器交换数据，也可以通过FTP发送推送通知、电子邮件和数据。

- 函数的Socket*组可以通过系统套接与远程主机建立一个TCP连接（包括安全的TLS）。操作原理很简单：创建一个套接，连接到服务器并开始读写数据。
- WebRequest函数专门用于处理网络资源，并可以轻松发送HTTP请求（包括GET和POST）。
- SendFTP，SendMail和 SendNotification是更加简单的函数，可以发送文件、电子邮件和移动设备通知。

为了保障终端用户的安全，可允许IP地址列表都在客户端实施。这个列表包含了允许MQL5程序通过Socket*和WebRequest函数连接的IP地址。例如，如果程序需要连接到https://www.someserver.com，那么该地址应由终端用户在列表中明确指出。不可以以编程的方式添加地址。




将一条明确消息添加到MQL5程序中，通知用户需要进行额外配置。您可以通过#property description，预警 或打印来完成。

| 函数 | 操作 |
| --- | --- |
| SocketCreate | 创建具有指定标识的套接并返回其句柄 |
| SocketClose | 关闭套接 |
| SocketConnect | 通过超时控制连接到服务器 |
| SocketIsConnected | 检查套接当前是否已连接 |
| SocketIsReadable | 获得可以从套接读取的字节数 |
| SocketIsWritable | 检查在当前时间是否可以将数据写入套接 |
| SocketTimeouts | 为系统套接对象设置数据接收和发送超时 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 操作 |
| --- | --- |
| SocketRead | 从套接读取数据 |
| SocketSend | 将数据写入套接 |
| SocketTlsHandshake | 通过TLS Handshake协议启动与指定主机的安全TLS (SSL)连接 |
| SocketTlsCertificate | 获得用于安全网络连接的证书数据 |
| SocketTlsRead | 读取来自安全TLS连接的数据 |
| SocketTlsReadAvailable | 读取来自安全TLS连接的所有可用数据 |
| SocketTlsSend | 通过安全TLS连接发送数据 |
| WebRequest | 将HTTP请求发送到一个指定服务器 |
| SendFTP | 将文件发送到FTP选项卡上指定的地址 |
| SendMail | 将电子邮件发送到选项窗口“电子邮件”选项卡中指定的地址 |
| SendNotification | 向“通知”选项卡中指定MetaQuotes ID的移动端发送推送通知 |

© 2000-2025, MetaQuotes Ltd.

# SocketCreate

创建具有指定标识的套接并返回其句柄。

```
int  SocketCreate(
   uint     flags          // 标识
   );
```

参数

*标识*

[in] 定义使用套接的模式的标识组合。目前，只支持一种标识 —SOCKET_DEFAULT。

返回值

如果成功创建套接，则会返回其句柄，否则 INVALID_HANDLE。

注释

若要从没有使用的套接释放计算机内存，请为其调用 SocketClose。

您可以从一个MQL5程序中创建最多128个套接。如果超出这个限制，错误号5271 (ERR_NETSOCKET_TOO_MANY_OPENED)会被写入 _LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
```

© 2000-2025, MetaQuotes Ltd.

```
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# SocketClose

关闭套接。

```
bool  SocketClose(
   const int  socket      // 套接句柄
   );
```

参数

*socket*

[in] 即将关闭的套接句柄。该句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

返回值

如果成功返回true，否则返回false。

注意

如果之前已为套接创建了通过SocketConnect的连接，则会中断连接。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
```

© 2000-2025, MetaQuotes Ltd.

```
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
```

© 2000-2025, MetaQuotes Ltd.

```
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# SocketConnect

通过超时控制连接到服务器。

```
bool  SocketConnect(
   int           socket,              // 套接
   const string  server,              // 连接地址
   uint          port,                // 连接端口
   uint          timeout_receive_ms   // 连接超时
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*server*

[in] 您要连接到的服务器的域名或服务器IP地址。

*port*

[in] 连接端口号。

*timeout_receive_ms*

[in] 连接超时，以毫秒计算。如果在此时间间隔内未建立连接，则停止尝试。

返回值

如果连接成功，返回true，否则返回false。

注意

连接地址应被添加到客户端（工具\选项\EA交易）可允许地址列表中。

如果连接失败，错误号5272 (ERR_NETSOCKET_CANNOT_CONNECT)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
```

© 2000-2025, MetaQuotes Ltd.

```
input int     Port   =80;
bool          ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
```

© 2000-2025, MetaQuotes Ltd.

```
            }
         }
      }
    while(GetTickCount()<timeout_check && !IsStopped());
    return(false);
   }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
```

© 2000-2025, MetaQuotes Ltd.

```
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# SocketIsConnected

检查套接字当前是否已连接。

```
bool  SocketIsConnected(
   const int  socket      // 套接字句柄
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate()函数返回。当不正确的句柄传递到_LastError时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)被激活。

返回值

如果套接字已连接，返回true，否则返回false。

注意

SocketIsConnected()函数可以检查当前套接字连接状态。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                              SocketIsConnected.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                           https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address    ="www.mql5.com";
input int    Port       =80;
input bool   CloseSocket=true;
bool         ExtTLS     =false;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 创建一个套接字并取得它的句柄
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果一切正常，就连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         PrintFormat("Established connection to %s:%d",Address,Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接有证书保护，显示它的数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:      ",subject);
            Print("   Issuer:     ",issuer);
            Print("   Number:     ",serial);
            Print("   Print:      ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送 GET 请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
        }
      //--- 如果设置了标志，就在使用后关闭套接字
      if(CloseSocket)
         SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());

//--- 检查服务器连接
   bool connected=SocketIsConnected(socket);
//--- 如果没有连接就终止运行
   if(!connected)
     {
      Print("No connection to server");
     }
//--- 如果服务器连接存在
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("Connection to the server is available\nThe connection needs to be closed.
       //--- 关闭套接字并再次检查连接状态
       SocketClose(socket);
       connected=SocketIsConnected(socket);
      }

//--- 显示当前服务器连接状态
   Print("Currently connected: ",(connected ? "opened" : "closed"));
   /*
   result in case CloseSocket = true:
   No connection to server
   Currently connected: closed

   result in case CloseSocket = false:
   Connection to the server is available
   The connection needs to be closed. Closing
   Currently connected: closed
   */
  }
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
//--- 把字符串转换成字符数组, 丢弃结尾的零
   char req[];
   int  len=StringToCharArray(request,req)-1;

   if(len<0)
      return(false);
//--- 如果使用了通过端口443的安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用了常规TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;
//--- 在有数据时从套接字读取数据，但不能超过超时时间
   do
     {
      uint len=SocketIsReadable(socket);
```

© 2000-2025, MetaQuotes Ltd.

```
      if(len)
        {
         int rsp_len;
         //--- 不同的读取命令取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout_ms);
         //--- 解析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 仅显示响应头
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
            //--- 更新读取超时过期时间
            timeout_check=GetTickCount64()+timeout_ms;
           }
        }
     }
   while(GetTickCount64()<timeout_check && !IsStopped());

   return(false);
  }
```

另 见

SocketConnect, SocketIsWritable, SocketCreate, SocketClose

© 2000-2025, MetaQuotes Ltd.

# SocketIsReadable

获得可以从套接读取的字节数。

```
uint  SocketIsReadable(
   const int  socket      // 套接句柄
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当不正确的句柄传递到_LastError时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)被激活。

返回值

可以读取的字节数。错误的情况下，返回0。

注意

如果执行函数时系统套接发生错误，那么通过SocketConnect建立的连接则中断。

调用SocketRead之前，请检查套接是否具有读取数据的功能。否则，如果没有数据，SocketRead函数则在延迟程序执行的timeout_ms内等待数据。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# SocketIsWritable

检查在当前时间是否可以将数据写入套接字。

```
bool  SocketIsWritable(
   const int  socket      // 套接字句柄
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

返回值

如果可以写入，返回true，否则返回false。

注意

这个函数允许您检查是否可以将数据立即写入套接字。

如果执行函数时系统套接字发生错误，那么通过SocketConnect建立的连接则中断。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                              SocketIsWritable.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address    ="www.mql5.com";
input int    Port       =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 创建一个套接字并取得它的句柄
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 如果一切正常，就连接
   if(SocketConnect(socket,Address,Port,1000))
     {
      PrintFormat("Established connection to %s:%d",Address,Port);

      string   subject,issuer,serial,thumbprint;
      datetime expiration;
      //--- 如果连接有证书保护，显示它的数据
      if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
        {
         Print("TLS certificate:");
         Print("   Owner:      ",subject);
         Print("   Issuer:     ",issuer);
         Print("   Number:     ",serial);
         Print("   Print:      ",thumbprint);
         Print("   Expiration: ",expiration);
         ExtTLS=true;
        }
      //--- 向服务器发送 GET 请求
      if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
        {
         Print("GET request sent");
         //--- 读取响应
         if(!HTTPRecv(socket,1000))
            Print("Failed to get a response, error ",GetLastError());
        }
      else
         Print("Failed to send GET request, error ",GetLastError());
     }
   else
     {
      PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
     }
   //--- 在日志中显示当前时刻向套接字写入数据的能力
   if(SocketIsWritable(socket))
      Print("At the current moment in time, writing data to the socket is possible"
   else
      Print("It is not possible to write data to the socket at the current time");
   //--- 使用后关闭套接字
   if(SocketClose(socket))
      Print("Now the socket is closed");
  }
else
   Print("Failed to create a socket, error ",GetLastError());
/*
结果：
 At the current moment in time, writing data to the socket is possible
 Socket is closed now
 */
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
//--- 把字符串转换成字符数组, 丢弃结尾的零
   char req[];
   int  len=StringToCharArray(request,req)-1;

   if(len<0)
      return(false);
//--- 如果使用了通过端口443的安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用了常规TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;
//--- 在有数据时从套接字读取数据，但不能超过超时时间
   do
     {
      uint len=SocketIsReadable(socket);

      if(len)
        {
         int rsp_len;
         //--- 不同的读取命令取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout_ms);
         //--- 解析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 仅显示响应头
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
              {
               Print("HTTP answer header received:");
```

```
                Print(StringSubstr(result,0,header_end));
                return(true);
               }
              //--- 更新读取超时过期时间
              timeout_check=GetTickCount64()+timeout_ms;
             }
           }
      }
   while(GetTickCount64()<timeout_check && !IsStopped());

   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SocketTimeouts

为系统套接字对象设置数据接收和发送超时。

```
bool  SocketTimeouts(
   int           socket,              // 套接字
   uint          timeout_send_ms,     // 数据发送超时
   uint          timeout_receive_ms   // 数据获取超时
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*timeout_send_ms*

[in] 数据发送超时，以毫秒计算。

*timeout_receive_ms*

[in] 数据获取超时，以毫秒计算。

返回值

如果成功返回true，否则返回false。

注意

当通过SocketRead读取数据时，请不要将系统对象超时与设置的对象超时混淆。在操作系统中，SocketTimeout为一个套接字对象设置一次超时。这些超时将应用于通过这个套接字读取和发送数据的所有函数。在SocketRead中，为某个数据读取操作设置超时。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                                SocketTimeouts.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version     "1.00"

#define   TIMEOUT_SEND     5000
#define   TIMEOUT_RECEIVE  5000
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建一个套接字并取得它的句柄
   int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- 为套接字系统对象设置接收和发送数据的超时
   if(SocketTimeouts(socket,TIMEOUT_SEND,TIMEOUT_RECEIVE))
      PrintFormat("timeouts were successfully set",socket,TIMEOUT_SEND,TIMEOUT_RECEIVE
   else
      PrintFormat("SocketTimeouts(%d, %I64d, %I64d) failed. Error %d",socket,TIMEOUT_S
//--- 关闭套接字
   SocketClose(socket);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SocketRead

从套接读取数据。

```
int  SocketRead(
   int           socket,              // 套接
   uchar&        buffer[],            // 用于从套接读取数据的缓冲区
   uint          buffer_maxlen,       // 即将读取的字节数
   uint          timeout_ms           // 读取超时
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当不正确的句柄传递到_LastError时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)被激活。

*buffer*

[out] 引用读取数据的uchar类型数组。动态数组大小随着读取字节数的增加而增加。数组大小不能超过INT_MAX (2147483647)。

*buffer_maxlen*

[in] 即将读取到*buffer[]*数组的字节数。不适合数组的数据保留在套接中。它们可以通过下一次调用SocketRead来接收。*buffer_maxlen*不能超过INT_MAX (2147483647)。

*timeout_ms*

[in] 数据读取超时，以毫秒计算。如果这个时间内没有获得数据，则会停止尝试，并且函数返回-1。

返回值

如果成功，则返回读取字节数。错误的话，返回-1。

注意

如果执行函数时系统套接发生错误，那么通过SocketConnect建立的连接则中断。

如果数据读取错误，错误号5273 (ERR_NETSOCKET_IO_ERROR)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs
```

© 2000-2025, MetaQuotes Ltd.

```
input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
```

© 2000-2025, MetaQuotes Ltd.

```
                return(true);
               }
            }
         }
      }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
         }
       //--- 用后关闭套接
       SocketClose(socket);
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

另 见

SocketTimeouts, MathSwap

© 2000-2025, MetaQuotes Ltd.

# SocketSend

将数据写入套接。

```
int  SocketSend(
   int           socket,              // 套接
   const uchar&  buffer[],            // 数据缓冲区
   uint          buffer_len           // 缓冲区大小
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*buffer*

[in] 通过即将发送至套接的数据，引用uchar类型数组。

*buffer_len*

[in] "缓冲区"数组大小。

返回值

如果成功，返回写入套接的字节数。错误的话，返回-1。

注意

如果执行函数时系统套接发生错误，那么通过SocketConnect建立的连接则中断。

如果数据写入错误，错误号5273 (ERR_NETSOCKET_IO_ERROR)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
```

© 2000-2025, MetaQuotes Ltd.

```
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

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

另 见

SocketTimeouts, MathSwap, StringToCharArray

© 2000-2025, MetaQuotes Ltd.

# SocketTlsHandshake

通过TLS Handshake协议启动与指定主机的安全TLS (SSL)连接。Handshake (交握)过程中，客户和服务器就连接参数达成一致：应用的协议版本和数据加密方法。

```
bool  SocketTlsHandshake(
   int           socket,              // 套接字
   const string  host                 // 主机地址
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*host*

[in] 建立安全连接的主机地址。

返回值

如果成功返回true，否则返回false。

注释

安全连接之前，该程序应该用SocketConnect与主机建立一个标准的TCP连接。

如果安全连接失败，错误号5274 (ERR_NETSOCKET_HANDSHAKE_FAILED)会被写入_LastError。

当连接到443端口时，不需要调用该函数。这是一个用于安全TLS(SSL)连接的标准的TCP端口。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                            SocketTlsHandshake.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   SERVER    "smtp.gmail.com"
#define   PORT      587
//+------------------------------------------------------------------+
//| 手动切换到安全连接                                                 |
//+------------------------------------------------------------------+
bool TlsHandshake(int socket)
  {
//--- 获取服务器问候语
```

```
   string rsp;

   if(!RecvString(socket,rsp))
      return(false);
//--- 问候服务器
   if(!SendString(socket,"EHLO my.domain.com\r\n"))
      return(false);
//--- 获取服务器响应，其中包含支持的命令列表
   if(!RecvString(socket,rsp))
      return(false);
//--- 打印问候语
   Print("SERVER: ",rsp);
//--- 通知服务器，我们想使用TLS从不安全连接切换到安全连接
   if(!SendString(socket,"STARTTLS\r\n"))
      return(false);
//--- 获取服务器响应
   if(!RecvString(socket,rsp))
      return(false);
//--- 在该示例中，我们不检查服务器关于切换到TLS的准备就绪（"准备启动TLS"）的响应

//--- 使用TLS握手协议启动与指定主机的安全TLS（SSL）连接
   if(SocketTlsHandshake(socket,InpTimeout))
      return(true);

   Print("SocketTlsHandshake() failed. Error ",GetLastError());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 创建一个套接字并取得它的句柄
   int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- 通过 PORT 端口连接到服务器 SERVER
   if(!SocketConnect(socket,SERVER,PORT,10000))
     {
      Print("SocketConnect() failed. Error ",GetLastError());
     }
   else
     {
      //--- 建立了非安全连接
      PrintFormat("%s connection has been established to %s:%d",(PORT==443 ? "A secure
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 切换到安全连接
      if(PORT!=443 && TlsHandshake(socket))
        {
         PrintFormat("Unsecured connection to %s:%d switched to secured",SERVER,PORT);
         //--- 如果连接有证书保护, 显示它的数据
         string   subject,issuer,serial,thumbprint;
         datetime expiration;

         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:      ",subject);
            Print("   Issuer:     ",issuer);
            Print("   Number:     ",serial);
            Print("   Print:      ",thumbprint);
            Print("   Expiration: ",expiration);
           }
        }
     }
//--- 在使用后关闭套接字
   SocketClose(socket);
   /*
   结果:
   An unsecured connection has been established to smtp.gmail.com:587
   SERVER: 220 smtp.gmail.com ESMTP a640c23a62f3a-a9b1f298319sm82305866b.105 - gsmtp

   SERVER: 250-smtp.gmail.com at your service, [37.193.40.122]
   250-SIZE 35882577
   250-8BITMIME
   250-STARTTLS
   250-ENHANCEDSTATUSCODES
   250-PIPELINING
   250-CHUNKING
   250 SMTPUTF8

   SERVER: 220 2.0.0 Ready to start TLS

   SocketTlsHandshake(): A secure connection to smtp.gmail.com:587 is now established
   TLS certificate:
      Owner:  /CN=smtp.gmail.com
      Issuer:  /C=US/O=Google Trust Services/CN=WR2
      Number:     1f:f4:db:2a:5a:e6:dc:52:0a:4c:05:ce:81:cc:c3:f7
      Print: d6be8af229b5329cd3d4c2789c02aa94f89b421c
      Expiration: 2024.12.30 08:25:30
   */
  }
//+------------------------------------------------------------------+
//| 向服务器发送字符串                                               |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool SendString(int socket,const string str)
  {
//--- 把字符串转换成字符数组
   uchar data[];
   int   size=StringToCharArray(str,data,0,str.Length(),CP_UTF8);
//--- 发送数据至套接字
   ResetLastError();
   if(SocketSend(socket,data,size)==size)
      return(true);
//--- 数据发送错误
   Print("Failed to send data to server. Error ",GetLastError());
   return false;
  }
//+------------------------------------------------------------------+
//| 从服务器获取字符串                                               |
//+------------------------------------------------------------------+
bool RecvString(int socket,string& result,uint timeout_ms=1000)
  {
//--- 等待数据出现在套接字上
   ulong wait_time_end=GetMicrosecondCount()+timeout_ms*1000;

   while(!SocketIsReadable(socket))
     {
      Sleep(10);
      //--- 等待数据超时-返回NULL作为响应
      if(wait_time_end<GetMicrosecondCount())
        {
         Print("ERROR: No response from server");
         return(false);
        }
     }
//--- 从套接字读取数据
   uchar  data[128];
   uint   size=0;
   string resp=NULL;

   do
     {
      uchar b[1];
      int   n=SocketRead(socket,b,1,1000);

      if(n < 0)
         break;

      if(n)
        {
         data[size++]=b[0];

         if(size==data.Size())
```

© 2000-2025, MetaQuotes Ltd.

```
            {
             resp += CharArrayToString(data,0,data.Size(),CP_UTF8);
             size = 0;
            }
         }
      }
   while(SocketIsReadable(socket));
//--- 将读取的数据复制到字符串中
   if(size)
      resp+=CharArrayToString(data,0,size,CP_UTF8);
//--- 如果字符串为空，则出现错误
   if(!resp.Length())
     {
      Print("ERROR: No response from server");
      return(false);
     }
//--- 返回字符串
   result=resp;
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SocketTlsCertificate

获得用于安全网络连接的证书数据。

```
bool  SocketTlsCertificate(
   int           socket,              // 套接
   string&       subject,             // 证书所有者
   string&       issuer,              // 证书发行者
   string&       serial,              // 证书序列号
   string&       thumbprint,          // 证书打印
   datetime&     expiration           // 证书到期
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*subject*

[in] 证书所有者名称。相对应 "主题"字段。

*issuer*

[in] 证书发行者名称。相对应 "发行者"字段。

*serial*

[in] 证书序列号。相对应 "SerialNumber"字段。

*thumbprint*

[in] 证书打印。相对应整个证书文件的SHA-1散列 (包括发行者签名的所有字段)。

*到期*

[in] datetime格式的许可证到期日期。

返回值

如果成功返回true，否则返回false。

注意

证书数据只能在使用SocketTlsHandshake建立一个安全连接之后来请求。

如果证书获取错误，错误号5275 (ERR_NETSOCKET_NO_CERTIFICATE)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
```

© 2000-2025, MetaQuotes Ltd.

```
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
            }
          }
      }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
```

© 2000-2025, MetaQuotes Ltd.

```
            Print("Failed to send GET request, error ",GetLastError());
         }
       else
         {
          Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
         }
       //--- 用后关闭套接
       SocketClose(socket);
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# SocketTlsRead

读取来自安全TLS连接的数据。

```
int  SocketTlsRead(
   int           socket,              // 套接
   uchar&        buffer[],            // 用于从套接读取数据的缓冲区
   uint          buffer_maxlen        // 即将读取的字节数
   );
```

参数

*socket*

[in] 套接句柄通过SocketCreate函数返回。当不正确的句柄传递到_LastError时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)被激活。

*buffer*

[out] 引用读取数据的uchar类型数组。动态数组大小随着读取字节数的增加而增加。数组大小不能超过INT_MAX (2147483647)。

*buffer_maxlen*

[in] 即将读取到*buffer[]*数组的字节数。不适合数组的数据保留在套接中。它们可以通过下一次调用SocketTLSRead来接收。*buffer_maxlen*不能超过INT_MAX (2147483647)。

返回值

如果成功，则返回读取字节数。错误的话，返回-1。

注意

如果执行函数时系统套接发生错误，那么通过SocketConnect建立的连接则中断。

直到接收到指定的数据量或达到超时 (SocketTimeouts)，才执行该函数。

如果数据读取错误，错误号5273 (ERR_NETSOCKET_IO_ERROR)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

例如：

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port    =80;
```

© 2000-2025, MetaQuotes Ltd.

```
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| 向服务器发送命令                                                 |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- 如果通过443端口使用安全TLS连接
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- 如果使用标准的TCP连接
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- 读取套接数据，直至套接数据不长过超时且仍然存在
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- 各种读取命令，取决于连接是否安全
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- 分析响应
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 只打印响应标题
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
```

© 2000-2025, MetaQuotes Ltd.

```
         }
      }
    while(GetTickCount()<timeout_check && !IsStopped());
    return(false);
   }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 一切顺利情况下继续连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接受到证书保护，则显示其数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- 向服务器发送GET请求
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- 用后关闭套接
      SocketClose(socket);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

另 见

SocketTimeouts, MathSwap

© 2000-2025, MetaQuotes Ltd.

# SocketTlsReadAvailable

读取来自安全TLS连接的所有可用数据。

```
int  SocketTlsReadAvailable(
   int           socket,               // 套接字
   uchar&        buffer[],             // 用于从套接字读取数据的缓冲区
   const uint    buffer_maxlen         // 即将读取的字节数
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate函数返回。当不正确的句柄传递到_LastError时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)被激活。

*buffer*

[out] 引用读取数据的uchar类型数组。动态数组大小随着读取字节数的增加而增加。数组大小不能超过INT_MAX (2147483647)。

*buffer_maxlen*

[in] 即将读取到buffer[]数组的字节数。不适合数组的数据保留在套接中。它们可以通过下一次调用SocketTlsReadAvailable或SocketTlsRead来接收。buffer_maxlen不能超过INT_MAX (2147483647)。

返回值

如果成功，则返回读取字节数。错误的话，返回-1。

注意

如果执行函数时系统套接字发生错误，那么通过SocketConnect建立的连接则中断。

如果数据读取错误，错误号5273 (ERR_NETSOCKET_IO_ERROR)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                      SocketTlsReadAvailable.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"
#property script_show_inputs
//+------------------------------------------------------------------+
//| 脚本输入参数                                                     |
//+------------------------------------------------------------------+
input string InpMethod ="GET";             // Method (HEAD,GET)
```

© 2000-2025, MetaQuotes Ltd.

```
input string InpServer ="www.google.com"; // Server
input uint   InpPort   =443;              // Port
input uint   InpTimeout=5000;             // Timeouts
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
   Print("Server: ",InpServer);
   Print("Port: ",InpPort);
//--- 创建一个套接字并取得它的句柄
   const int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- 为套接字系统对象设置接收和发送数据的超时
   if(!SocketTimeouts(socket,InpTimeout,InpTimeout))
     {
      PrintFormat("SocketTimeouts(%u, %u) failed. Error %d",InpTimeout,InpTimeout,GetL
      SocketClose(socket);
      return;
     }
//--- 通过端口连接到服务器
   if(!SocketConnect(socket,InpServer,InpPort,InpTimeout))
     {
      PrintFormat("SocketConnect('%s', %u, %u) failed. Error %d",InpServer,InpPort,Inp
      SocketClose(socket);
      return;
     }
//--- 获取用于保护网络连接的证书数据
   string   subject,issuer,serial,thumbprint;
   datetime expiration;

   if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
     {
      Print("TLS certificate:");
      Print("   Owner:      ",subject);
      Print("   Issuer:     ",issuer);
      Print("   Number:     ",serial);
      Print("   Print:      ",thumbprint);
      Print("   Expiration: ",expiration);
     }
   else
     {
      //--- 服务器不提供证书-报告不安全的连接并离开
      Print("The connection is not secured by a certificate");
```

© 2000-2025, MetaQuotes Ltd.

```
      SocketClose(socket);
      return;
     }
//--- 向服务器发送请求
   string request=StringFormat("%s / HTTP/1.1\r\nHost: %s\r\nUser-Agent: MetaTrader 5\

   if(HTTPSendTLS(socket,request))
     {
      //--- 请求已发送-获取响应
      Print("\nRequest sent. Starting page loading...");
      uchar response[]; // all received data (document header and body)

      if(!HTTPRecvTLS(socket,response,InpTimeout))
        {
         Print("There were errors while reading the page");
         SocketClose(socket);
         return;
        }
      //--- 报告接收到的数据字节数
      PrintFormat("%u bytes received",response.Size());
      //--- 仅显示获取的页头部
      string result    =CharArrayToString(response,0,WHOLE_ARRAY,CP_UTF8);
      int    header_end=StringFind(result,"\r\n\r\n");

      if(header_end>0)
        {
         Print("\nHTTP answer header received:");
         Print(StringSubstr(result,0,header_end));
        }
     }
//--- 在使用后关闭套接字
   SocketClose(socket);
   /*
   结果：
   Server: www.google.com
   Port: 443
   TLS certificate:
      Owner: /CN=www.google.com
      Issuer: /C=US/O=Google Trust Services/CN=WR2
      Number: 0d:43:b1:4a:bb:9c:15:96:10:e1:3d:55:23:9f:25:4e
      Print: 89167618e5017f813aff981c88ce422dc1016bdf
      Expiration: 2024.12.30 08:26:35

   Request sent. Starting page loading...
   HTTPRecvTLS: Document received within 27 attempts
   25185 bytes received

   HTTP answer header received:
   HTTP/1.1 200 OK
```

© 2000-2025, MetaQuotes Ltd.

```
   Date: Fri, 25 Oct 2024 17:12:42 GMT
   Expires: -1
   Cache-Control: private, max-age=0
   Content-Type: text/html; charset=ISO-8859-1
   Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src '
   Accept-CH: Sec-CH-Prefers-Color-Scheme
   P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
   Server: gws
   X-XSS-Protection: 0
   X-Frame-Options: SAMEORIGIN
   Set-Cookie: AEC=AVYB7coyYMCdweTDTaWeGYzmRnxzKGqsOEosH_VkbCn8xhWkFz6v0kxQFw; expires
   Set-Cookie: NID=518=J02X02Ff4v_9sMcNoUz-1SolmuG08E26Gs438ik0J_SOJUMy7of-P-qup-LaNSW
   Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
   Accept-Ranges: none
   Vary: Accept-Encoding
   Transfer-Encoding: chunked
   */
  }
//+------------------------------------------------------------------+
//| 通过安全连接发送HTTP请求                                          |
//+------------------------------------------------------------------+
bool HTTPSendTLS(int socket,const string request)
  {
//--- 把字符串转换成字符数组, 丢弃结尾的零
   char req[];
   int  len=StringToCharArray(request,req,0,WHOLE_ARRAY,CP_UTF8)-1;

   if(len<0)
      return false;

   return(SocketTlsSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| 通过安全连接获取网页                                              |
//+------------------------------------------------------------------+
bool HTTPRecvTLS(int socket,uchar &response[],const uint timeout_ms)
  {
//--- 在超时到期之前从安全TLS连接读取可用数据
   ulong timeout_check=GetTickCount64()+timeout_ms;
   uchar block[1024];   // 用于从套接字读取块数据的缓冲区
   uint  attempt=0;     // 请求的数据块数量
   int   err    =0;     // 错误代码

   ResetLastError();

   do
     {
      //--- 读入块,最大1024字节
      int len=SocketTlsReadAvailable(socket,block,1024);
```

© 2000-2025, MetaQuotes Ltd.

```
      if(len>0)
        {
         attempt++;
         //--- 合并获得的数据块
         ArrayCopy(response,block,response.Size());
         //--- 分析获得的数据，定义头部、页主体、结尾或加载错误等。
         //...
         //...
         //...
         timeout_check=GetTickCount64()+timeout_ms;
        }
      else
         Sleep(10);

      err=GetLastError();
     }
   while(!IsStopped() && GetTickCount()<timeout_check && !err);
//--- 读取时有错误吗？
   if(err)
     {
      Print("Error ",err);
      return(false);
     }

   PrintFormat("%s: Document received within %d attempts",__FUNCTION__,attempt);
   return(true);
  }
```

另 见

SocketTimeouts, MathSwap

© 2000-2025, MetaQuotes Ltd.

# SocketTlsSend

通过安全TLS连接发送数据。

```
int  SocketTlsSend(
   int           socket,              // 套接字
   const uchar&  buffer[],            // 数据缓冲区
   uint          buffer_len           // 缓冲区大小
   );
```

参数

*socket*

[in] 套接字句柄通过SocketCreate函数返回。当传递不正确的句柄时，错误号5270 (ERR_NETSOCKET_INVALIDHANDLE)会被写入_LastError。

*buffer*

[in] 通过即将发送的数据，引用uchar类型数组。

*buffer_len*

[in] "缓冲区"数组大小。

返回值

如果成功，返回写入套接的字节数。错误的话，返回-1。

注意

如果执行函数时系统套接字发生错误，那么通过SocketConnect建立的连接则中断。

如果数据写入错误，错误号5273 (ERR_NETSOCKET_IO_ERROR)会被写入_LastError。

这个函数只能从EA交易和脚本中调用，因为它们在自己的执行线程中运行。如果从指标调用，GetLastError()则返回4014错误号 - "函数不允许调用"。

示例：

```
//+------------------------------------------------------------------+
//|                                                SocketTlsSend.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd.   |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port    =443;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnStart(void)
  {
   int socket=SocketCreate();
//--- 检查句柄
   if(socket!=INVALID_HANDLE)
     {
      //--- 如果一切正常，就连接
      if(SocketConnect(socket,Address,Port,1000))
        {
         PrintFormat("Established connection to %s:%d",Address,Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- 如果连接有证书保护，显示它的数据
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:       ",subject);
            Print("   Issuer:      ",issuer);
            Print("   Number:      ",serial);
            Print("   Print:       ",thumbprint);
            Print("   Expiration: ",expiration);
           }
         //--- 向服务器发送 GET 请求
         string request="GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r\n\r
         char   req[];
         int    len=StringToCharArray(request,req)-1;

         if(len<0)
           {
            Print("StringToCharArray() failed. Error ", GetLastError());
            SocketClose(socket);
            return;
           }
         //--- 如果使用了通过端口443的安全TLS连接
         if(SocketTlsSend(socket,req,len)==len)
           {
            Print("GET request sent");
            //--- 读取响应
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 使用后关闭套接字
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
//| 读取服务器响应                                                   |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
//--- 在有数据时从套接字读取数据，但不能超过超时时间
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;

   do
     {
      uint len=SocketIsReadable(socket);

      if(len)
        {
         //--- 读取并解析安全TLS连接的数据
         int rsp_len=SocketTlsRead(socket,rsp,len);

         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- 仅显示响应头
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
            //--- 更新读取超时过期时间
            timeout_check=GetTickCount64()+timeout_ms;
           }
        }
     }
   while(GetTickCount64()<timeout_check && !IsStopped());

   return(false);
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

SocketTimeouts, MathSwap, StringToCharArray

© 2000-2025, MetaQuotes Ltd.

# WebRequest

发送 HTTP 请求到指定服务器。该函数有两个版本：

1. 发送简单请求 使用内容-类型标题的"key=value"类型：application/x-www-form-urlencoded

```
int  WebRequest(
   const string      method,           // HTTP 请求方式
   const string      url,              // url 地址
   const string      cookie,           // cookie
   const string      referer,          // 参照页
   int               timeout,          // 超时
   const char        &data[],          // HTTP请求信息数组
   int               data_size,        // data[] 数组大小
   char              &result[],        // 服务器响应数据数组
   string            &result_headers   // 服务器响应标头
   );
```

2. 发送任何类型的请求 为了各种网络服务的更灵活互动指定自定义的标题设置。

```
int  WebRequest(
   const string      method,           // HTTP 方法
   const string      url,              // URL
   const string      headers,          // 标题
   int               timeout,          // 超时
   const char        &data[],          // HTTP 信息主体的数组
   char              &result[],        // 包含服务器反应数据的数组
   string            &result_headers   // 服务器响应标题
   );
```

参数

*method*

[in] HTTP 请求方法。

*url*

[in] URL 地址。

*headers*

[in] "key: value"类型的请求标题，以分行符 "\r\n"分隔。

*cookie*

[in] Cookie。

*referer*

[in] HTTP 请求标头的推荐页字段

*timeout*

[in] 超时时间以毫秒为单位。

*data[]*

[in]HTTP 请求信息数组。

© 2000-2025, MetaQuotes Ltd.

*data_size*

[in] data[] 数组大小。

*result[]*

[out] 服务器响应数据数组。

*result_headers*

[out] 服务器响应标头。

返回值

服务器响应的HTTP状态码，或错误情况下的 -1 。

注意

若要能够使用WebRequest 函数，您必须在 "选项"窗口 "EA交易"标签的允许URLs列表中添加一个URL。

WebRequest 是一个同步函数。这意味着它会阻止程序执行并且等候请求服务器的响应。当收到发送请求的响应时，延迟至关重要，这就是禁止从指标调用该函数的原因，因为指标按照交易品种组指标的自己的执行线程操作。在一个图表上执行指标延迟可能导致该交易品种的所有图表停止更新。

该函数只能从专家和脚本调用，因为他们按照自己的执行线程操作。从指标调用，GetLastError() 函数将返回错误4014 -"函数不允许调用"。

WebRequest() 函数不能在 策略测试中执行。

例如：

```
void OnStart()
  {
   string cookie=NULL,headers;
   char   post[],result[];
   string url="https://finance.yahoo.com";
//--- To enable access to the server, you should add URL "https://finance.yahoo.com"
//--- to the list of allowed URLs (Main Menu->Tools->Options, tab "Expert Advisors"):
//--- Resetting the last error code
   ResetLastError();
//--- Downloading a html page from Yahoo Finance
   int res=WebRequest("GET",url,cookie,NULL,500,post,0,result,headers);
   if(res==-1)
     {
      Print("Error in WebRequest. Error code  =",GetLastError());
      //--- Perhaps the URL is not listed, display a message about the necessity to ad
      MessageBox("Add the address '"+url+"' to the list of allowed URLs on tab 'Expert
     }
   else
     {
      if(res==200)
        {
         //--- Successful download
         PrintFormat("The file has been successfully downloaded, File size %d byte.",A
```

© 2000-2025, MetaQuotes Ltd.

```
         //PrintFormat("Server headers: %s",headers);
         //--- Saving the data to a file
         int filehandle=FileOpen("url.htm",FILE_WRITE|FILE_BIN);
         if(filehandle!=INVALID_HANDLE)
           {
            //--- Saving the contents of the result[] array to a file
            FileWriteArray(filehandle,result,0,ArraySize(result));
            //--- Closing the file
            FileClose(filehandle);
           }
         else
            Print("Error in FileOpen. Error code =",GetLastError());
        }
      else
         PrintFormat("Downloading '%s' failed, error code %d",url,res);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# SendFTP

在地址中发送文件，指定在Publisher窗口中。

```
bool  SendFTP(
   string  filename,           // 通过ftp发送的文件
   string  ftp_path=NULL       // ftp服务器上传的文件
   );
```

参量

*filename*

[in] 发送文件名称

*ftp_path=NULL*

[in] FTP目录，如果目录未标明，直接在使用设置中描述

返回值

错误返回是“false”。

注释

发送文件应该在 *terminal_directory\MQL5\files* 或子文件夹中保存，如果在设置中未标明FTP地址或者访问密码，无法发送。

SendFTP() 函数在策略测试中不工作。

示例：

```
//+------------------------------------------------------------------+
//|                                                      SendFTP.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   FILENAME  "SomeFile.bin"
#define   PATH      NULL
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 检查从终端中发送文件至 FTP 地址的许可
   if(!TerminalInfoInteger(TERMINAL_FTP_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send messages to t
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 发送文件
   ResetLastError();
   if(!SendFTP(FILENAME, PATH))
      Print("SendFTP() failed. Error ",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# SendMail

在Mailbox建立的窗口中发送指定地址邮件。

```
bool  SendMail(
   string  subject,        // 标题
   string  some_text       // email 文本
   );
```

参量

*subject*

[in] 邮件标题

*some_text*

[in] 邮件主题

返回值

如果邮件发送到派送队列则true，否则是false

注释

发送在设置中能被禁止，邮件地址可能未标明，查看错误信息调用 GetLastError()

SendMail() 函数在策略测试中不工作。

示例：

```
//+------------------------------------------------------------------+
//|                                                     SendMail.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   SUBJECT   "Test SendMail"
#define   TEXT      "Text for SendMail() function"
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 检查从终端中发送电子邮件的许可
   if(!TerminalInfoInteger(TERMINAL_EMAIL_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send email message
      return;
     }
//--- 发送邮件
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
   if(!SendMail(SUBJECT, TEXT))
      Print("SendMail() failed. Error ",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# SendNotification

发送推送通知到在 "通知"标签指定MetaQuotes ID的移动程序端。

```
bool  SendNotification(
   string  text          // 通知文本
   );
```

参数

*text*

[in] 通知文本。信息长度不应该超过 255 字符。

返回值

如果通知从程序端成功发送，则返回true；如果失败则返回false。推送通知失败后检查时，GetLastError () 可能返回其中以下一种错误：

- 4515 - ERR_NOTIFICATION_SEND_FAILED,
- 4516 - ERR_NOTIFICATION_WRONG_PARAMETER,
- 4517 - ERR_NOTIFICATION_WRONG_SETTINGS,
- 4518 - ERR_NOTIFICATION_TOO_FREQUENT.

注意

为SendNotification()函数设置了严格的使用限制：每秒钟调用不能超过2次，每分钟调用不能超过10次。动态监控使用频率。如果违反限制，函数将被禁用。

SendNotification() 函数在策略测试中不工作。

示例：

```
//+------------------------------------------------------------------+
//|                                             SendNotification.mq5 |
//|                                Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   MESSAGE   "Test Message"
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- 检查从终端中发送通知的许可
   if(!TerminalInfoInteger(TERMINAL_NOTIFICATIONS_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send notifications
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 发送通知
   ResetLastError();
   if(!SendNotification(MESSAGE))
      Print("SendNotification() failed. Error ",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# 程序端全局变量

有一组为全局变量工作的函数设置。

客户端的全局变量不能与MQL5程序中已声明的全局范围变量混淆

自从上一次访问全局变量保存在客户端四周，然后就自动删除。访问全局变量不仅要设置新值，还有访问全局变量值。

客户端的全局变量容易同时从MQL5程序加载到客户端。

| 函数 | 功能 |
| --- | --- |
| GlobalVariableCheck | 检测带有指定名的全局变量的存在性 |
| GlobalVariableTime | 返回上次访问全局变量的时间 |
| GlobalVariableDel | 删除全局变量 |
| GlobalVariableGet | 返回全局变量值 |
| GlobalVariableName | 全局变量列表中依据序列号返回全局变量名 |
| GlobalVariableSet | 为全局变量设新值 |
| GlobalVariablesFlush | 强制在磁盘 保存全局变量内容 |
| GlobalVariableTemp | 给全局变量设新值，只在程序端当前状态存在 |
| GlobalVariableSetOnCondition | 根据状态设置现存全局变量的新值 |
| GlobalVariablesDeleteAll | 删除指定前缀的全局变量 |
| GlobalVariablesTotal | 返回全局变量总数 |

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableCheck

检测带有指定名称的全局变量的存在性。

```
bool  GlobalVariableCheck(
   string  name      // 全局变量名称
   );
```

参量

*name*

[in] 全球变量名称

返回值

如果全局变量存在，返回真值，否则返回错误值。

全局变量在客户端保存自最后使用的四周，然后自动删除。

另见

GlobalVariableTime()

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableTime

当全局变量最后访问时返回时间。

```
datetime  GlobalVariableTime(
   string  name      // 名称
   );
```

参量

*name*

[in] 全局变量名称。

返回值

函数返回上次访问的指定全局变量，调用变量获得值也需要考虑是否访问，为了获得错误细节，调用 GetLastError() 函数。

注释

全局变量在客户端保存自最后使用的四周，然后自动删除。

另见

GlobalVariableCheck()

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableDel

从客户端中删除全局变量。

```
bool  GlobalVariableDel(
   string  name      // 全局变量名称
   );
```

参量

*name*

[in] 全局变量名称。

返回值

如果成功，函数返回真值，否则返回错误值。为获得有关错误信息，有必要调用GetLastError()函数。

注释

全局变量在客户端保存自最后使用的四周，然后自动删除。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableGet

返回客户端存在的全局变量存在值，有2个变量函数可以使用。

1. 立即返回全局变量值。

```
double  GlobalVariableGet(
   string  name      // 全局变量名称
   );
```

2. 返回真值或者错误值取决于函数是否成功运行。如果成功，客户端全局变量通过引用将变量传递到第二参量中。

```
bool  GlobalVariableGet(
   string  name,              // 全局变量名称
   double& double_var         //该变量包括全局变量值
   );
```

参量

*name*

[in]全局变量名称。

*double_var*

[out] 双精度函数的变量目标，接收存储在客户端的全局变量的值。

返回值

存在的全局变量值或者 error是0，了解更多关于误差值，调用 GetLastError()

注释

全局变量在客户端保存自最后使用的四周，然后自动删除。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableName

通过序列号返回全局变量名称。

```
string  GlobalVariableName(
   int  index      // 全局变量列表中全局变量数
   );
```

参量

*index*

[in] 全局变量列表中的序号，应该大于或等于0，小于 GlobalVariablesTotal()函数。

返回值

全局变量的名称通过序列号在全局变量列表中体现出来，更多关于误差的细节，调用 GetLastError()。

注释

全局变量在客户端保存自最后使用的四周，然后自动删除。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableSet

为全局变量设置新值，如果变量不存在，系统增添新的全局变量。

```
datetime  GlobalVariableSet(
   string  name,       // 全局变量名称
   double  value       // 设置值
   );
```

参量

*name*

[in] 全局变量名称。

*value*

[in] 新数值。

返回值

如果成功，函数返回上一修正时间，否则返回0，更多关于 错误的细节，调用 GetLastError()。

注释

全局变量名称不应超过63个字符。全局变量在客户端保存自最后使用的四周，然后自动删除。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariablesFlush

强制存储所有全局变量信息到磁盘。

```
void  GlobalVariablesFlush();
```

返回值

无返回值。

注释

当工作结束，终端编辑所有全局变量，但数据会在计算机操作失败时突然丢失。该函数允许独立控制存储全局变量过程，以防发生偶然事故。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableTemp

函数试图建立暂时的全局变量，如果变量不存在，系统增设一个新的暂时全局变量。

```
bool  GlobalVariableTemp(
   string  name      // 全局变量名称
   );
```

参量

*name*

[in] 暂时全局变量名称。

返回值

如果成功，函数返回真值，否则是错误值，更多关于 错误的细节，调用 GetLastError()。

注释

暂时的全局变量只在客户端运行时存在，在客户端停止工作时自动删除。执行 GlobalVariablesFlush() 暂时全局变量时不编辑到磁盘里。

建立暂时全局变量以后，可以像 客户端全局变量一样访问并修改。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariableSetOnCondition

如果当前值等于第三参量check_value，设置现有全局变量的新值。如果没有全局变量，函数将生成错误ERR_GLOBALVARIABLE_NOT_FOUND (4501)并且返回false。

```
bool  GlobalVariableSetOnCondition(
   string  name,             // 全局变量名称
   double  value,            // 条件是true下的变量新值
   double  check_value       // 检测值的状态
   );
```

参量

*name*

[in] 全局变量名称。

*value*

[in] 新值。

*check_value*

[in] 检测当前全局变量值。

返回值

如果成功，函数返回真值，否则返回错误值。更多关于 error 的细节，调用 GetLastError()。如果当前全局变量值不同于check_value，函数返回false。

注释

函数将为全局变量提供自动通道，所以它可以提供一个信号量在同一个客户终端内同时被几个EA调用。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariablesDeleteAll

删除客户端的全局变量。

```
int  GlobalVariablesDeleteAll(
   string     prefix_name=NULL,     // 所有带有前缀的全局变量
   datetime   limit_data=0          // 所有该日期前已更改的全局变量
   );
```

参量

*prefix_name=NULL*

[in]将被删除的全局变量的命名前缀。如果指定前缀是NULL或者空字符串，所有满足数据标准的变量都会被删除。

*limit_data=0*

[in] 通过上次修正时间来选择全局变量，函数删除在此日期之前改变的全局变量，如果参量是0，所有满足第一标准 (前缀)的变量都会被删除。

返回值

删除变量数量。

Note

如果每个选项等于0 (prefix_name = NULL and limit_data = 0)，函数删除终端所有的全局变量。如果所有的参量都是制定的，删除与该参量相关的全局变量。

全局变量在客户端保存自最后使用的四周，然后自动删除。

© 2000-2025, MetaQuotes Ltd.

# GlobalVariablesTotal

返回客户端所有全局变量的数量。

```
int  GlobalVariablesTotal();
```

返回值

全局变量数量。

注释

全局变量在客户端保存自最后使用的四周，然后自动删除。调用全局变量不仅需要设立新值，也需要读取全局变量的值。

© 2000-2025, MetaQuotes Ltd.

# 文件函数

这是一组文档工作函数。

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

有两个目录（附带子目录）放置工作文档：

- terminal_data_folder\MQL5\FILES\（终端菜单中选择浏览"文件"-"打开数据目录"）；
- 安装在电脑中所有终端的普通文件夹-通常在C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\Files目录中。

获得目录名称的程序方式，使用TerminalInfoString()函数，使用ENUM_TERMINAL_INFO_STRING 表达式：

```
//--- 存储程序端数据的文件夹
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
//--- 客户端常用文件夹
   string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

禁止从其他目录使用文件。

文件函数允许使用所谓的"命名管道"。为此，只需用合适的参数调用FileOpen()函数。

| 函数 | 功能 |
|---|---|
| FileSelectDialog | 创建一个文件或文件夹打开/创建对话框 |
| FileFindFirst | 依照指定过滤器启动目录中的文件搜索 |
| FileFindNext | 继续通过FileFindFirst()函数启动的搜索 |
| FileFindClose | 结束搜索程序 |
| FileOpen | 打开指定名字和标记的文件 |
| FileDelete | 删除指定文件 |
| FileFlush | 将输入/输出文件缓冲区的所有数据写入磁盘 |
| FileGetInteger | 获得文件的整型属性 |
| FileIsEnding | 读取过程中定义文件末端 |
| FileIsLineEnding | 读取过程中定义文本行末端 |
| FileClose | 关闭之前打开的文件 |
| FileIsExist | 检测文件存在性 |
| FileCopy | 从本地或者共享文件夹复制原文件到另一个文件 |
| FileMove | 移动或者重命名文件 |
| FileReadArray | 读取除了BIN类型文件字符串外的任何类型数组 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 |
| --- | --- |
| FileReadBool | 读取CSV类型文件当前位置字符串直到定界符（或者直到文本行末端）和转换读取的字符串到布尔型值 |
| FileReadDatetime | 读取CSV类型文件以"YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" or "HH:MI:SS"格式字符串并且转换它到日期时间值 |
| FileReadDouble | 从文件指针当前位置读取双精度值 |
| FileReadFloat | 从文件指针当前位置读取浮点值 |
| FileReadInteger | 从文件指针当前位置读取整型，短整型或者字符型值 |
| FileReadLong | 从文件指针当前位置读取一个长整型值 |
| FileReadNumber | 从CSV类型文件读取当前位置字符串直到定界符（或者直到文本行末端）并且转换读取字符串到双精度值 |
| FileReadString | 在文件中文件指针当前位置读取字符串 |
| FileReadStruct | 文件指针当前位置读取二进制文件内容到作为参量传递的结构 |
| FileSeek | 根据指定位置有关的指定二进制数移动文件指针位置 |
| FileSize | 返回相应打开文件大小 |
| FileTell | 返回相应打开文件的文件指针的当前位置 |
| FileWrite | 向CSV或者TXT类型文件写入数据 |
| FileWriteArray | 写入字符串以外任何类型数组到BIN类型文件 |
| FileWriteDouble | 从文件指针当前位置写入双精度型值到二进制文件 |
| FileWriteFloat | 从文件指针当前位置写入浮点型值到二进制文件 |
| FileWriteInteger | 从文件指针当前位置写入整型值到二进制文件 |
| FileWriteLong | 从文件指针当前位置写入长整型值到二进制文件 |
| FileWriteString | 从文件指针当前位置写入字符串参量值到BIN或者TXT文件 |
| FileWriteStruct | 从文件指针当前位置写入作为参量传递的结构到二进制文件 |
| FileLoad | 指定二进制文件的所有数据读入数值类型或简单结构的传递数组 |
| FileSave | 将传递数组所有元素写入二进制文件作为参数 |
| FolderCreate | 在文件目录创建一个文件夹 |
| FolderDelete | 清除选定目录。如果文件夹不是空的，那么无法清除。 |
| FolderClean | 删除指定文件夹中所有文件 |

如果文件打开使用 FileOpen()函数编写，那么路径中所有指定的子文件夹将会创建。

© 2000-2025, MetaQuotes Ltd.

# FileSelectDialog

创建一个文件或文件夹打开/创建对话框。

```
int  FileSelectDialog(
   string   caption,              // 窗口标题
   string   initial_dir,          // 初始目录
   string   filter,               // 扩展过滤器
   uint     flags,                // 标识组合
   string&  filenames[],          // 包含文件名的数组
   string   default_filename      //默认文件名
   );
```

参数

*caption*

[in] 对话框窗口标题。

*initial_dir*

[in] 相对于MQL5\Files的初始目录名，其内容将显示在对话框中。如果该值为NULL，对话框中显示MQL5\Files。

*filter*

[in] 将要显示在所选对话框窗口的文件的扩展过滤器。其他格式的文件被隐藏。

*flags*

[in]定义对话框窗口模式的标识组合。标识定义如下：
FSD_WRITE_FILE -打开文件对话框；
FSD_SELECT_FOLDER -只允许选择文件夹；
FSD_ALLOW_MULTISELECT -允许选择多个文件；
FSD_FILE_MUST_EXIST -所选文件应该存在；
FSD_COMMON_FOLDER -文件位于所以客户端的常规文件夹\Terminal\Common\Files。

*filenames[]*

[out] 所选文件/文件夹的名称所在的字符串数组。

*default_filename*

[in] 默认文件/文件夹名称。如果已指定，则在测试时自动将名称添加到打开的对话框中，并返回到*filenames[]*数组。

返回值

如果成功完成，该函数将返回文件名称可在*filenames[]*中获得的所选文件的数量。如果用户在不选择文件的情况下关闭对话框，则函数返回0。如果执行不成功，则返回值小于0。使用GetLastError()可以获得错误代码。

注意

出于安全考虑，在MQL5语言中严格管控文件的处理。通过MQL5语言在文件操作中使用的文件不能位于文件沙箱之外（即MQL5\Files目录外）。

© 2000-2025, MetaQuotes Ltd.

在客户端的MQL5\Files（或在测试情况下的testing_agent_directory\MQL5\Files）目录中搜索*initial_dir*名称。如果在标识中设置了FSD_COMMON_FOLDER，则对初始目录的搜索在所有客户端的常规文件夹中执行\Terminal\Common\Files。

*过滤器*参数表示有效的文件，应该在"<description 1>|<extension 1>|<description 2>|<extension 2>..."格式中设置，例如，“文本文件(*.txt)|*.txt|所有文件(*.*)|*.*”。选择的第一个扩展“文本文件(*.txt)|*.txt”作为默认文件类型。

如果*过滤器*=NULL，则对话框窗口中所选文件的掩码为“所有文件(*.*)|*.*|”

如果设置了*default_filename*参数，则调用FileSelectDialog()返回1，而在非可视化测试期间*default_filename*值本身会复制到*filenames[]*数组中。

该函数不可用于自定义指标，因为FileSelectDialog()调用在等待用户响应时，会一直暂停执行线程。由于每个交易品种的所有指标都在一个线程中执行，因此暂停会造成无法操作该交易品种的所有时间周期的所有图表。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 从客户端的通用文件夹获取要下载的文本文件的名称
   string filenames[];
   if(FileSelectDialog("Select files to download", NULL,
                       "Text files (*.txt)|*.txt|All files (*.*)|*.*",
                       FSD_ALLOW_MULTISELECT|FSD_COMMON_FOLDER, filenames, "data.txt")
     {
      //--- 显示每个选定文件的名称
      int total=ArraySize(filenames);
      for(int i=0; i<total; i++)
         Print(i, ": ", filenames[i]);
     }
   else
     {
     Print("Files not selected");
     }
//---
  }
```

另见

FileOpen、FileIsExist、FileDelete、FileMove、FolderCreate、FolderDelete、FolderClean、文件开始标签

© 2000-2025, MetaQuotes Ltd.

# FileFindFirst

该函数根据指定过滤器启动搜索一个目录下的文件或子目录。

```
long  FileFindFirst(
   const string   file_filter,           // 字符串 - 搜寻过滤器
   string&        returned_filename,     // 找到的文件名或子目录名
   int            common_flag=0          // 定义搜索
   );
```

参量

*file_filter*

[in] 搜索过滤器。与\Files目录相关联的子目录（或者嵌套在子目录里的序列），需要寻找文件的地方，指定在过滤器中

*returned_filename*

[out] 返回参数，这里，如果成功，则设置第一个找到的文件或子目录名称。仅返回文件名（包括扩展名），无论它们是否在搜索过滤器中指定，都不会包括目录和子目录。

*common_flag*

[in] 标记 决定文件位置，如果common_flag = FILE_COMMON,文件为所有客户端放置共享文件夹\Terminal\Common\Files。否则，文件放置在本地文件夹中。

返回值

返回被搜索物件的句柄，它应该用于通过FileFindNext()函数进一步分类文件和子目录，返回搜索物件的句柄，它应该用于在没有关于过滤器的文件和子目录时（尤其是目录为空时），通过函数或INVALID_HANDLE进一步分类文件和子目录。搜索之后，句柄必须使用FileFindClose()函数来关闭。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 显示启动脚本时的导入参量的窗口
#property script_show_inputs
//--- filter
input string InpFilter="Dir1\\*";
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   string int_dir="";
   int    i=1,pos=0,last_pos=-1;
//--- search for the last backslash
   while(!IsStopped())
     {
      pos=StringFind(InpFilter,"\\",pos+1);
      if(pos>=0)
         last_pos=pos;
      else
         break;
     }
//--- 过滤器包含文件夹名称
   if(last_pos>=0)
      int_dir=StringSubstr(InpFilter,0,last_pos+1);
//--- get the search handle in the root of the local folder
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- 检查 FileFindFirst() 是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 检查循环中传递的字符串是否是文件或目录名称
      do
        {
         ResetLastError();
         //--- 如果这是一个文件，函数返回true，如果是目录，它返回错误 ERR_FILE_IS_DIRECTORY
         FileIsExist(int_dir+file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "Di
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索句柄
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

另 见

FileFindNext, FileFindClose

© 2000-2025, MetaQuotes Ltd.

# FileFindNext

函数继续通过 FileFindFirst()搜索起始位置。

```
bool  FileFindNext(
   long      search_handle,          // 搜索处理程序
   string&   returned_filename       // 找到的文件或子目录的名称
   );
```

参量

*search_handle*

[in] 搜索处理，通过 FileFindFirst()恢复。

*returned_filename*

[out] 找到的下一个文件或子目录的名称。仅返回文件名(包括扩展名)，无论它们是否在搜索过滤器中指定，都不会包括目录和子目录。

返回值

如果成功，返回真值，否则返回错误值。

示例：

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 过滤器
input string InpFilter="*";
//+------------------------------------------------------------------+
//| 脚本程序启动开始                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- 在本地文件夹根目录下接收搜索句柄
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- 检查FileFindFirst()函数是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 检查循环中传递的字符串是否是文件或目录名称
      do
        {
         ResetLastError();
         //--- 如果是文件，函数将返回true，如果是目录，函数将生成错误ERR_FILE_IS_DIRECTORY。
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "Di
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索句柄
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

FileFindFirst, FileFindClose

© 2000-2025, MetaQuotes Ltd.

# FileFindClose

函数关闭搜索处理。

```
void  FileFindClose(
   long  search_handle      //  搜索处理程序
   );
```

参量

*search_handle*

[in] 搜索处理，通过 FileFindFirst()恢复。

返回值

无值返回。

注释

函数必须调用到空出来的系统资源。

示例：

```
//--- 启动脚本时启动输入参数的窗口
#property script_show_inputs
//--- 过滤器
input string InpFilter="*";
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- 在本地文件夹根目录下接收搜索句柄
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- 检查FileFindFirst()函数是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 检查循环中传递的字符串是否是文件或目录名称
      do
        {
         ResetLastError();
         //--- 如果是文件，函数将返回true，如果是目录，函数将生成错误5018。
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File"
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索句柄
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

另见

FileFindFirst, FileFindNext

© 2000-2025, MetaQuotes Ltd.

# FileIsExist

检测文件是否存在。

```
bool  FileIsExist(
   const string  file_name,        // 文件名
   int           common_flag=0     // 搜索区
   );
```

参量

*file_name*

[in] 被检测的文件名

*common_flag=0*

[in] 标记 决定文件位置，如果common_flag = FILE_COMMON,文件为所有客户端放置共享文件夹\Terminal\Common\Files。否则，文件放置在本地文件夹中。

返回值

如果指定文件存在返回真值。

注释

检测的文件可以证明是子目录。这种情况下，FileIsExist()函数将返回false，_LastError变量将会登录错误5018 - "这是一个目录，而不是文件"(请见FileFindFirst 函数示例)。

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实 施文件操作的文件意味着，不能在文件沙箱外。

如果common_flag = FILE_COMMON,函数为所有客户端在共享文件中寻找文件\Terminal\Common\Files，否则函数在本地文件夹中寻找文件 (MQL5 \ Files or MQL5 \ Tester \ Files检测状态下)。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 旧文件的日期
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;       // 存储文件名称的变量
   string   filter="*.txt"; // 搜索文件的过滤器
   datetime create_date;     // 文件创建日期
   string   files[];         // 文件名称列表
   int      def_size=25;     // 默认数组大小
   int      size=0;          // 文件名称
//--- 为数组分配内存
   ArrayResize(files,def_size);
//--- 在本地文件夹根目录下接收搜索句柄
   long search_handle=FileFindFirst(filter,file_name);
//--- 检查FileFindFirst()函数是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 在循环中搜索文件
      do
        {
         files[size]=file_name;
         //--- 增加数组大小
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- 重置错误值
         ResetLastError();
         //--- 接收文件创建日期
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- 检查文件是否是旧的
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- 删除旧文件
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索句柄
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- 检查已保留哪些文件
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
```

© 2000-2025, MetaQuotes Ltd.

```
            PrintFormat("%s file deleted!",files[i]);
        }
    }
```

另 见

FileFindFirst

© 2000-2025, MetaQuotes Ltd.

# FileOpen

函数打开文件的指定名称和标签。

```
int  FileOpen(
   string  file_name,           // 文件名
   int     open_flags,          // 标记的组合
   short   delimiter='\t',      // 定界符
   uint    codepage=CP_ACP      // 代码页
   );
```

参量

*file_name*

[in] 文件名称包括子文件夹，如果文件为编辑开放，就会建立子文件夹。

*open_flags*

[in] 标记的组合 决定文件夹的操作方式，标签如下定义
FILE_READ为阅读展开
FILE_WRITE 为编辑展开
FILE_BIN 二次编辑阅读模式 (不会从字符串到字符串转换)
FILE_CSV csv类型文件 (所有记录项目转换成统一码字符串或者ansi类型，被定界符分开)
FILE_TXT 简单文本文件 (与csv相同，但不考虑定界符)
FILE_ANSIANSI类型线 (单一字节交易品种)
FILE_UNICODE UNICODE类型线 (双精度函数字符)
FILE_SHARE_READ 从各自程序中共享阅读
FILE_SHARE_WRITE 从各自程序中共享编辑
FILE_COMMON 所有客户端的共享文件夹中的文件位置 \Terminal\Common\Files.

*delimiter='\t'*

[in] 在txt或者csv文件中使用分隔符的值，如果csv文件标识符不是指定的，默认到标签，如果txt文件标识符不是制定的，就不使用分隔符。如果分隔符清晰建立成0，不使用分隔符。

*codepage=CP_ACP*

[in] 代码页的值，最多使用 代码页 提供占用常量。

返回值

如果文件成功打开，函数返回文件处理，然后访问文件数据，失败后返回 INVALID_HANDLE。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实 施文件操作的文件意味着，不能在文件沙箱外。

如果文件应该以特定的编码方式 (指定包含代码页值的codepage参数)读取，请确保设置FILE_ANSI标识。如果没有指定的FILE_ANSI标识，则文本文件将以Unicode形式读取，而无需进行任何转换。

文件在客户端子文件 MQL5\Files (或者testing_agent_directory\MQL5\Files)中打开，如果FILE_COMMON在制定标签中，文件在所有MT5客户端的共享文件中打开。

"命名管道"可以根据以下规则打开：

© 2000-2025, MetaQuotes Ltd.

- 管道名称是一个字符串，应该有以下显示："\\servername\pipe\pipename"，这里servername - 网络中的服务器名称，而pipename是管道名称。如果管道用在同一个计算机上，服务器名称可以忽略，但有一个点应该被插入以替代它："\\.\pipe\pipename"。尝试连接管道的客户应该知道其名称。
- FileFlush() 和 FileSeek() 在从管道读取和写入的顺序操作之间应该在文件开始被调用。

特殊符号'\'应该用于显示字符串。因此，'\' 应该在MQL5应用程序编写名称时，使用双杠。这意味着上面示例应该显示如下代码："\\\\servername\\pipe\\pipename"。

使用命名管道的更多信息可以查看文章"通过MetaTrader5沟通使用命名管道，而非DLLs"

示例：

```
//+------------------------------------------------------------------+
//| 启动函数的脚本程序                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 错误的打开文件方法
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   string filename=terminal_data_path+"\\MQL5\\Files\\"+"fractals.csv";
   int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
   if(filehandle<0)
     {
      Print("Failed to open the file by the absolute path ");
      Print("Error code ",GetLastError());
     }
//---  “文件沙盒效应”中正确的工作方法
   ResetLastError();
   filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
   if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(_Period));
      FileClose(filehandle);
      Print("FileOpen OK");
     }
   else Print("Operation FileOpen failed, error ",GetLastError());
//---在MQL5\Files\中创建附着目录的另一个示例
   string subfolder="Research";
   filehandle=FileOpen(subfolder+"\\fractals.txt",FILE_WRITE|FILE_CSV);
      if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(_Period));
      FileClose(filehandle);
      Print("The file must be created in the folder "+terminal_data_path+"\\"+subfolde
     }
   else Print("File open failed, error ",GetLastError());
  }
```

另见

FileFindFirst, 创建文件夹, 打开文件标记

© 2000-2025, MetaQuotes Ltd.

# FileClose

通过 FileOpen()关闭先前打开的文件。

```
void  FileClose(
   int  file_handle      // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

返回值

无值返回。

示例：

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="file.txt";     // 文件名称
input string InpDirectoryName="Data";    // 目录名称
input int    InpEncodingType=FILE_ANSI; // ANSI=32 或 UNICODE=64
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打印文件将要使用的路径
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
//--- 重置错误值
   ResetLastError();
//--- 打开阅读文件(如果文件不存在,错误将产生)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      //--- 打印文件内容
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# FileCopy

函数从本地或者共享文件中复制原始文件到另一文件中。

```
bool  FileCopy(
   const string  src_file_name,      // 源文件名称
   int           common_flag,        // 位置
   const string  dst_file_name,      // 目标文件名称
   int           mode_flags          // 访问模式
   );
```

参量

*src_file_name*

[in] 复制的文件名称。

*common_flag*

[in] 标记 决定文件位置，如果common_flag = FILE_COMMON,文件为所有客户端放置共享文件夹\Terminal\Common\Files。否则，文件放置在本地文件夹中 (例如，*common_flag=0*)。

*dst_file_name*

[in] 结果文件名。

*mode_flags*

[in] 访问标记。参量只包括2个标签：FILE_REWRITE 和/或 FILE_COMMON -忽略其他标签。如果文件已经存在，FILE_REWRITE标签未指定，文件就不能再编辑，函数返回错误值。

返回值

若失败函数返回false。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

如果新文件已经存在，复制文件在mode_flags参量中取决于FILE_REWRITE标签的可用性。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpSrc="source.txt";        // 源
input string InpDst="destination.txt";   // 复制
input int    InpEncodingType=FILE_ANSI; // ANSI=32 或 UNICODE=64
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 显示源内容 它必须存在)
   if(!FileDisplay(InpSrc))
      return;
//--- 检查复制文件是否已经存在(可能没有创建)
   if(!FileDisplay(InpDst))
     {
      //--- 复制文件不存在,无FILE_REWRITE标识复制 (正确复制)
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
   else
     {
      //--- 复制文件已经存在,尝试无FILE_REWRITE标识复制(不正确复制)
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
      //--- InpDst 文件的内容保持不变
      FileDisplay(InpDst);
      //--- 再次使用FILE_REWRITE标识复制(如果文件存在,正确复制)
      if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
//--- 接收InpSrc文件复制
   FileDisplay(InpDst);
  }
//+------------------------------------------------------------------+
//| 阅读文件内容                                                     |
//+------------------------------------------------------------------+
bool FileDisplay(const string file_name)
  {
//--- 重置错误值
   ResetLastError();
//--- 打开文件
   int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- 在循环中显示文件内容
      Print("+----------------------+");
      PrintFormat("File name = %s",file_name);
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      Print("+----------------------+");
      //--- 关闭文件
      FileClose(file_handle);
      return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 打开文件失败
   PrintFormat("%s is not opened, error = %d",file_name,GetLastError());
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# FileDelete

在客户端中删除本地文件夹的指定文件。

```
bool  FileDelete(
   const string  file_name,      // 要删除的文件名
   int           common_flag=0   // 要删除的文件位置
   );
```

参量

*file_name*

[in] 文件名称。

*common_flag=0*

[in] 标记 决定文件位置，如果common_flag = FILE_COMMON,文件为所有客户端放置共享文件夹\Terminal\Common\Files。否则，文件放置在本地文件夹中。

返回值

如果失败函数返回 false。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实 施文件操作的文件意味着，不能在文件沙箱外。

删除客户端中本地文件夹中的指定文件(MQL5\files or MQL5\tester\files in case of testing). 如果common_flag = FILE_COMMON，函数就会从所有客户端的共享文件中删除文件。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 旧文件的日期
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // 存储文件名称的变量
   string   filter="*.txt"; // 搜索文件的过滤器
   datetime create_date;    // 文件创建日期
   string   files[];        // 文件名称列表
   int      def_size=25;    // 默认数组大小
   int      size=0;         // 文件名称
//--- 为数组分配内存
   ArrayResize(files,def_size);
//--- 在本地文件夹根目录下接收搜索句柄
   long search_handle=FileFindFirst(filter,file_name);
//--- 检查FileFindFirst()函数是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 在循环中搜索文件
      do
        {
         files[size]=file_name;
         //--- 增加数组大小
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- 重置错误值
         ResetLastError();
         //--- 接收文件创建日期
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- 检查文件是否是旧的
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- 删除旧文件
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索句柄
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- 检查已保留哪些文件
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
```

© 2000-2025, MetaQuotes Ltd.

```
            PrintFormat("%s file deleted!",files[i]);
        }
    }
```

© 2000-2025, MetaQuotes Ltd.

# FileMove

从本地或共享文件夹中移动文件到另一文件夹。

```
bool  FileMove(
   const string  src_file_name,     // 进行移动操作的文件名
   int           common_flag,       // 位置
   const string  dst_file_name,     // 目标文件名
   int           mode_flags         // 访问模式
   );
```

参量

*src_file_name*

[in] 移动/重命名的文件名称。

*common_flag*

[in] 标记 决定文件位置，如果common_flag = FILE_COMMON,文件为所有客户端放置共享文件夹\Terminal\Common\Files。否则，文件放置在本地文件夹中。

*dst_file_name*

[in] 操作后的文件名称。

*mode_flags*

[in] 访问标记。参量只包括2个标签：FILE_READ和/或者 FILE_COMMON -忽略另外标签。如果文件已经存在并且FILE_READ标签未指定，文件不会再编辑，函数返回错误值。

返回值

如果失败，函数返回错误值。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

如果新文件已经存在，复制文件在mode_flags参量中取决于FILE_READ标签的可用性。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpSrcName="data.txt";
input string InpDstName="newdata.txt";
input string InpSrcDirectory="SomeFolder";
input string InpDstDirectory="OtherFolder";
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string local=TerminalInfoString(TERMINAL_DATA_PATH);
   string common=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- 接收文件路径
   string src_path;
   string dst_path;
   StringConcatenate(src_path,InpSrcDirectory,"//",InpSrcName);
   StringConcatenate(dst_path,InpDstDirectory,"//",InpDstName);
//--- 检查源文件是否存在 如果不 - 退出）
   if(FileIsExist(src_path))
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpSrcName,local,InpSrc
   else
     {
      PrintFormat("Error, %s source file not found",InpSrcName);
      return;
     }
//--- 检查结果文件是否已经存在
   if(FileIsExist(dst_path,FILE_COMMON))
     {
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpDstName,common,InpDs
      //--- 如果文件存在，移动应该通过FILE_REWRITE标识来完成
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON|FILE_REWRITE))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
   else
     {
      PrintFormat("%s file does not exist in the %s\\Files\\%s folder",InpDstName,comm
      //--- 如果文件不存在，移动应该无需通过FILE_REWRITE标识来完成
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
//--- 文件被移动;我们去看看
   if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
      Print("Success!");
   else
      Print("Error!");
  }
```

另 见

FileIsExist

© 2000-2025, MetaQuotes Ltd.

# FileFlush

在输入/输出文件缓冲区中编辑所有保留数据到磁盘。

```
void  FileFlush(
   int  file_handle      // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

返回值

没有返回值。

注释

当写入文件时，可能只有一段时间后才能实际上发现数据。要即时在文件中保存数据，请使用FileFlush()函数。如果没有使用函数，磁盘中还未保存的数据部分，将被迫只能在使用FileClose()函数关闭文件时写入。

函数应该在编写数据具有特定值时使用。应该记住，频繁的函数调用可能影响程序操作速度。

函数FileFlush ()必须在阅读和编辑文件操作之间调用。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 编写的文件名称
input string InpFileName="example.csv"; // 文件名
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 重置错误值
   ResetLastError();
//--- 打开文件
   int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- 写入文件数据
      for(int i=0;i<1000;i++)
        {
         //--- 调用编写函数
         FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYMBOL_BID),Sym
         //--- 每128次迭代时，在磁盘中保存数据
         if((i & 127)==127)
           {
            //--- 现在，数据将位于文件并且不会丢失，以防严重的错误。
            FileFlush(file_handle);
            PrintFormat("i = %d, OK",i);
           }
         //--- 0.01 秒暂停
         Sleep(10);
        }
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

另 见

FileClose

© 2000-2025, MetaQuotes Ltd.

# FileGetInteger

获得文件的整型属性。该函数有两个变量。

1. 通过文件句柄获得属性。

```
long  FileGetInteger(
   int                          file_handle,    // 文件句柄
   ENUM_FILE_PROPERTY_INTEGER   property_id     // 属性ID
   );
```

2. 通过文件名获得属性。

```
long  FileGetInteger(
   const string                 file_name,              // 文件名
   ENUM_FILE_PROPERTY_INTEGER   property_id,            // 属性 ID
   bool                         common_folder=false     // 文件在本地文件夹查看(false)
   );                                                   // 或在所有程序端的普通文件夹(true)
```

参量

*file_handle*

[in] 通过FileOpen()返回的文件夹说明符。

*file_name*

[in] 文件名。

*property_id*

[in] 文件属性ID。该值可以是ENUM_FILE_PROPERTY_INTEGER枚举值的一个。如果使用函数的第二变量，您只能收到以下属性的值：FILE_EXISTS, FILE_CREATE_DATE, FILE_MODIFY_DATE, FILE_ACCESS_DATE 和 FILE_SIZE。

*common_folder=false*

[in] 文件位置点。如果参数是false，则查看程序端数据文件夹。否则假定文件在所有共享文件夹terminals\Terminal\Common\Files (FILE_COMMON)。

返回值

属性的值。如果错误，则返回-1。要获得错误代码，请使用GetLastError()函数。

如果在通过名称获得属性时指定文件夹，函数无论如何都将有5018错误(ERR_MQL_FILE_IS_DIRECTORY)，虽然返回值可能是正确的。

注释

函数经常改变错误代码。如果成功完成，错误代码重置为NULL。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string path=InpDirectoryName+"//"+InpFileName;
   long   l=0;
//--- 打开文件
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_CSV);
   if(handle!=INVALID_HANDLE)
     {
      //--- 打印有关文件的所有信息
      Print(InpFileName," file info:");
      FileInfo(handle,FILE_EXISTS,l,"bool");
      FileInfo(handle,FILE_CREATE_DATE,l,"date");
      FileInfo(handle,FILE_MODIFY_DATE,l,"date");
      FileInfo(handle,FILE_ACCESS_DATE,l,"date");
      FileInfo(handle,FILE_SIZE,l,"other");
      FileInfo(handle,FILE_POSITION,l,"other");
      FileInfo(handle,FILE_END,l,"bool");
      FileInfo(handle,FILE_IS_COMMON,l,"bool");
      FileInfo(handle,FILE_IS_TEXT,l,"bool");
      FileInfo(handle,FILE_IS_BINARY,l,"bool");
      FileInfo(handle,FILE_IS_CSV,l,"bool");
      FileInfo(handle,FILE_IS_ANSI,l,"bool");
      FileInfo(handle,FILE_IS_READABLE,l,"bool");
      FileInfo(handle,FILE_IS_WRITABLE,l,"bool");
      //--- 关闭文件
      FileClose(handle);
     }
   else
      PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetLastError());
  }
//+------------------------------------------------------------------+
//| 显示文件属性的值                                                 |
//+------------------------------------------------------------------+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
  {
//--- 接收属性值
   ResetLastError();
   if((l=FileGetInteger(handle,id))!=-1)
     {
      //--- 接收的值，以正确格式显示
      if(!StringCompare(type,"bool"))
         Print(EnumToString(id)," = ",l ? "true" : "false");
      if(!StringCompare(type,"date"))
         Print(EnumToString(id)," = ",(datetime)l);
      if(!StringCompare(type,"other"))
         Print(EnumToString(id)," = ",l);
     }
   else
      Print("Error, Code = ",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

另见

文件操作，文件属性

© 2000-2025, MetaQuotes Ltd.

# FileIsEnding

在阅读过程中编辑文件末尾。

```
bool  FileIsEnding(
   int  file_handle      // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件标识符。

返回值

如果编辑或移动文件指标过程中能够到达文件末尾，函数返回真值。

注释

要定义文件结尾，该函数尝试从这里读取下一个字符串。如果字符串不存在，函数返回true，否则返回false。

示例：

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="file.txt";     // 文件名称
input string InpDirectoryName="Data";    // 目录名称
input int    InpEncodingType=FILE_ANSI; // ANSI=32 或 UNICODE=64
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打印文件将要使用的路径
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
//--- 重置错误值
   ResetLastError();
//--- 打开阅读文件(如果文件不存在，错误将产生)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      //--- 打印文件内容
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

© 2000-2025, MetaQuotes Ltd.

# FileIsLineEnding

在阅读过程中定义文本文件的末尾路线。

```
bool  FileIsLineEnding(
   int  file_handle      // 文件句柄
   );
```

参量

*file_handle*

[in] 文件说明符通过 FileOpen()返回。

返回值

如果阅读txt文件或者csv文件过程中到达路线末尾，返回真值（字符CR-LF）。

示例（执行FileWriteString 函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                        Demo_FileIsLineEnding.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- 读取数据的参数
input string InpFileName="RSI.csv";   // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 指标缓冲区
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- 买超变量
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
//--- 卖超变量
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 默认数组大小的变量
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- 为数组分配内存
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      double value;
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 读取字符串的第一个值
         value=FileReadNumber(file_handle);
         //--- 根据函数结果读取不同的数组
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
```

© 2000-2025, MetaQuotes Ltd.

```
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- 绑定数组
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 读取文件的字符串数据                                             |
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- 阅读直至到达字符串或文件的末尾
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- 阅读数量后转变这里
      if(flag)
         FileReadNumber(file_handle);
      //--- 存储当前日期
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- 必要时增加数组大小
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- 滑过第一迭代
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
```

© 2000-2025, MetaQuotes Ltd.

```
{
```

© 2000-2025, MetaQuotes Ltd.

```
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- 如果日期一致，柱形在超买区域
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - 红色
               color_buff[i]=0;
               //--- 增加计数器
               ovb_ind=j+1;
               break;
              }
           }
      //--- 检查是否任何数据仍然存在
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- 如果日期一致，柱形在超卖区域
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - 蓝色
               color_buff[i]=1;
               //--- 增加计数器
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- 返回prev_calculated值，以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| ChartEvent 事件处理程序                                          |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
```

© 2000-2025, MetaQuotes Ltd.

```
                     )
   {
//--- 根据比例更改指标宽度
    if(ChartGetInteger(0,CHART_SCALE)>3)
       PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
    else
       PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
   }
```

另 见

FileWriteString

© 2000-2025, MetaQuotes Ltd.

# FileReadArray

从BIN类型数组的除字符串类型外任意类型中阅读文件（或许是结构数组，不包括字符串，和动态数组）。

```
uint  FileReadArray(
   int     file_handle,              // 文件句柄
   void&   array[],                  // 要记录的数组
   int     start=0,                  // 开始编写数组位置
   int     count=WHOLE_ARRAY         // 读取计数
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*array[]*

[out] 数据可以加载到的数组。

*start=0*

[in] 书写数组的开始仓位。

*count=WHOLE_ARRAY*

[in] 阅读元素的数量。默认，阅读整个数组 (count=WHOLE_ARRAY)。

返回值

阅读元素的数量。

注释

字符串数组智能从TXT类型文件中阅读，如果必要的话，函数试着增加数组大小。

示例（执行FileWriteArray函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| 存储价格数据的结构                                               |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // 日期
   double            bid;  // 卖价
   double            ask;  // 买价
  };
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 结构数组
   prices arr[];
//--- 文件路径
   string path=InpDirectoryName+"//"+InpFileName;
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- 读取来自数组文件的所有数据
      FileReadArray(file_handle,arr);
      //--- 接收数组大小
      int size=ArraySize(arr);
      //--- 打印数组数据
      for(int i=0;i<size;i++)
         Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i].ask);
      Print("Total data = ",size);
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
      Print("File open failed, error ",GetLastError());
  }
```

另见

变量, FileWriteArray

© 2000-2025, MetaQuotes Ltd.

# FileReadBool

从当前位置CSV类型字符串中读取文件到定界符（或者到文本路线的末尾）。

```
bool  FileReadBool(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

返回值

线型阅读可以建立"true", "false"或者整数符号表示 "0" 或者 "1". 非零值转换成逻辑真值，函数返回修正值。

示例（执行FileWrite 函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                              Demo_FileReadBool.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//---- 图 Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- 图 Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- 读取数据的参数
input string InpFileName="MACD.csv";  // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 全局变量
int      ind=0;        // 指数
double   upbuff[];     // 向上箭头的指标缓冲区
double   downbuff[];   // 向下箭头的指标缓冲区
bool     sign_buff[]; // 信号数组 (true - 买入, false - 卖出)
datetime time_buff[]; // 信号到达时间数组
int      size=0;       // 信号数组大小
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- 首先,阅读信号的数量
      size=(int)FileReadNumber(file_handle);
      //--- 为数组分配内存
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- 读取文件数据
      for(int i=0;i<size;i++)
        {
         //--- 信号时间
         time_buff[i]=FileReadDatetime(file_handle);
         //--- 信号价值
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
       return(INIT_FAILED);
      }
//--- 绑定数组
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- 设置PLOT_ARROW中绘制的交易品种代码
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      upbuff[i]=0;
      downbuff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               //--- 根据信号绘制箭头
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另 见

布尔类型, FileWrite

© 2000-2025, MetaQuotes Ltd.

# FileReadDatetime

从CSV类型文件中读取字符串文件格式"YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD"或者"HH:MI:SS"转换到日期时间值类型。

```
datetime  FileReadDatetime(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in]通过 FileOpen()返回文件说明符。

返回值

日期时间类型值。

示例（执行 FileWrite 函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                      Demo_FileReadDateTime.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//---- 图 Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- 图 Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- 读取数据的参数
input string InpFileName="MACD.csv";  // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 全局变量
int      ind=0;        // 指数
double   upbuff[];     // 向上箭头的指标缓冲区
double   downbuff[];   // 向下箭头的指标缓冲区
bool     sign_buff[]; // 信号数组 (true - 买入,false - 卖出)
datetime time_buff[]; // 信号到达时间数组
int      size=0;       // 信号数组大小
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- 首先,阅读信号的数量
      size=(int)FileReadNumber(file_handle);
      //--- 为数组分配内存
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- 读取文件数据
      for(int i=0;i<size;i++)
        {
         //--- 信号时间
         time_buff[i]=FileReadDatetime(file_handle);
         //--- 信号价值
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- 关闭文件
      FileClose(file_handle);
     }
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
       return(INIT_FAILED);
      }
//--- 绑定数组
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- 设置PLOT_ARROW中绘制的交易品种代码
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      upbuff[i]=0;
      downbuff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               //--- 根据信号绘制箭头
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另见

日期时间类型, 字符串到时间, 时间到字符串, FileWrite

© 2000-2025, MetaQuotes Ltd.

# FileReadDouble

从当前二进制文件位置阅读双精度浮点编号 (双精度)。

```
double  FileReadDouble(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

返回值

双精度类型值。

注释

关于误差的更多细节，调用 GetLastError()。

示例 (执行 FileWriteDouble 函数示例后获得的文件在这里使用)

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- 读取数据的参数
input string InpFileName="MA.csv";     // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 全局变量
int      ind=0;
int      size=0;
double   ma_buff[];
datetime time_buff[];
//--- 指标缓冲区
double   buff[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 首先，阅读文件中的数据量
      size=(int)FileReadDouble(file_handle);
      //--- 为数组分配内存
      ArrayResize(ma_buff,size);
      ArrayResize(time_buff,size);
      //--- 读取文件数据
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadDouble(file_handle);
         ma_buff[i]=FileReadDouble(file_handle);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- 绑定数组和指数为0的指标缓冲区
```

© 2000-2025, MetaQuotes Ltd.

```
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               buff[i]=ma_buff[j];
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另见

真实型（双精度型，浮点型），字符串到双精度，双精度到字符串，FileWriteDouble

© 2000-2025, MetaQuotes Ltd.

# FileReadFloat

从当前二进制文件位置阅读双单精度浮点编号 (浮点)。

```
float  FileReadFloat(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符

返回值

浮点类型值。

注释

更多关于误差细节，调用 GetLastError().

示例 (执行 FileWriteFloat 函数示例后获得的文件在这里使用)

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                            Demo_FileReadFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "CloseLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- 读取数据的参数
input string InpFileName="Close.bin"; // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 全局变量
int      ind=0;
int      size=0;
double   close_buff[];
datetime time_buff[];
//--- 指标缓冲区
double   buff[];
double   color_buff[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- 为数组分配内存
   ArrayResize(close_buff,def_size);
   ArrayResize(time_buff,def_size);
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 阅读时间和价格的值
         time_buff[size]=(datetime)FileReadDouble(file_handle);
         close_buff[size]=(double)FileReadFloat(file_handle);
         size++;
         //--- 如果超过增加数组大小
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(close_buff,def_size);
            ArrayResize(time_buff,def_size);
           }
        }
      //--- 关闭文件
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
      return(INIT_FAILED);
     }
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,buff,INDICATOR_DATA);
   SetIndexBuffer(1,color_buff,INDICATOR_COLOR_INDEX);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      buff[i]=0;
      color_buff[i]=0; // 默认为红色
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               //--- 接收价格
               buff[i]=close_buff[j];
               //--- 如果当前价格超过之前的价格，为蓝色
               if(buff[i-1]>buff[i])
                  color_buff[i]=1;
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另 见

© 2000-2025, MetaQuotes Ltd.

真实型（双精度型，浮点型）, FileReadDouble, FileWriteFloat

© 2000-2025, MetaQuotes Ltd.

# FileReadInteger

从当前文件指示位置，函数读取整型，短型或图表型值，取决于规定字节长度。

```
int  FileReadInteger(
   int  file_handle,          // 文件句柄
   int  size=INT_VALUE        // 字节中整数的大小
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回的文件说明符。

*size=INT_VALUE*

[in] 应该阅读的字节数 (最多到4)。提供对应的常量：CHAR_VALUE = 1, SHORT_VALUE = 2 和 INT_VALUE = 4，所以函数可以阅读字符型，短整型或整型的整个值。

返回值

整型的值。该函数的值必须明确计算目标类型，例如您需要阅读的数据类型。因为返回整型的值，它可以轻松地转到任何整型值。文件指针就是阅读字节数的移动装置。

注释

当阅读少于4字节时，接收的结果始终是正数。如果阅读一个或两个字节，数字符号可以通过明确计算字符型 (1个字节)或短整型 (2个字节)来确定。获得3字节数字的符号并不重要，因为没有相应的基础类型。

示例 (执行FileWriteInteger函数示例后获得的文件在这里使用)

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 读取数据的参数
input string InpFileName="Trend.bin"; // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 全局变量
int      ind=0;
int      size=0;
datetime time_buff[];
//--- 指标缓冲区
double   buff[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- 为数组分配内存
   ArrayResize(time_buff,def_size);
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 附加变量
      int    arr_size;
      uchar  arr[];
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 找出多少交易品种用于编写时间
         arr_size=FileReadInteger(file_handle,INT_VALUE);
         ArrayResize(arr,arr_size);
         for(int i=0;i<arr_size;i++)
            arr[i]=(char)FileReadInteger(file_handle,CHAR_VALUE);
         //--- 存储时间价值
         time_buff[size]=StringToTime(CharArrayToString(arr));
         size++;
         //--- 如果超过增加数组大小
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(time_buff,def_size);
           }
```

© 2000-2025, MetaQuotes Ltd.

```
         }
       //--- 关闭文件
       FileClose(file_handle);
       PrintFormat("Data is read, %s file is closed",InpFileName);
      }
    else
      {
       PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
       return(INIT_FAILED);
      }
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(close,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               //--- 接收价格
               buff[i]=close[i];
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另 见

© 2000-2025, MetaQuotes Ltd.

整数到字符串, 字符串到整数, 整数类型, FileWriteInteger

© 2000-2025, MetaQuotes Ltd.

# FileReadLong

从二进制文件当前位置函数读取长型整数（8字节）。

```
long  FileReadLong(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过FileOpen()返回文件说明符。

返回值

长整型值。

示例（执行FileWriteLong函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                              Demo_FileReadLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "Volume"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrYellow
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- 读取数据的参数
input string InpFileName="Volume.bin"; // 文件名称
input string InpDirectoryName="Data";  // 目录名称
//--- 全局变量
int      ind=0;
int      size=0;
long     volume_buff[];
datetime time_buff[];
//--- 指标缓冲区
double   buff[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 首先，阅读文件中的数据量
      size=(int)FileReadLong(file_handle);
      //--- 为数组分配内存
      ArrayResize(volume_buff,size);
      ArrayResize(time_buff,size);
      //--- 读取文件数据
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadLong(file_handle);
         volume_buff[i]=FileReadLong(file_handle);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- 绑定数组和指数为0的指标缓冲区
   SetIndexBuffer(0,buff,INDICATOR_DATA);
```

© 2000-2025, MetaQuotes Ltd.

```
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==time_buff[j])
              {
               buff[i]=(double)volume_buff[j];
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated值，以便下次调用
   return(rates_total);
  }
```

另 见

整数类型, FileReadInteger, FileWriteLong

© 2000-2025, MetaQuotes Ltd.

# FileReadNumber

函数从CSV文件字符串中读取当前位置分隔符（或者直到文本字符串末尾）并转变阅读字符串到双精度类型值。

```
double  FileReadNumber(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过FileOpen()返回文件描述符。

返回值

双精度型值。

示例（执行FileWriteString函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadNumber.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- 读取数据的参数
input string InpFileName="RSI.csv";   // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//--- 指标缓冲区
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- 超买变量
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
//--- 超卖变量
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 默认数组大小的变量
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- 为数组分配内存
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      double value;
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 读取字符串的第一个值
         value=FileReadNumber(file_handle);
         //--- 根据函数结果读取不同的数组
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
```

© 2000-2025, MetaQuotes Ltd.

```
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- 绑定数组
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
//---- 设置图表上看不到的指标值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 读取文件的字符串数据                                             |
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- 阅读直至到达字符串或文件的末尾
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- 阅读数量后转变这里
      if(flag)
         FileReadNumber(file_handle);
      //--- 存储当前日期
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- 必要时增加数组大小
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- 滑过第一迭代
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
```

© 2000-2025, MetaQuotes Ltd.

```
{
```

© 2000-2025, MetaQuotes Ltd.

```
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- 还未处理的柱的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- 如果日期一致，柱形在超买区域
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - 红色
               color_buff[i]=0;
               //--- 增加计数器
               ovb_ind=j+1;
               break;
              }
           }
      //--- 检查是否任何数据仍然存在
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- 如果日期一致，柱形在超卖区域
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - 蓝色
               color_buff[i]=1;
               //--- 增加计数器
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- 返回prev_calculated值，以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| ChartEvent 事件处理程序                                          |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
```

© 2000-2025, MetaQuotes Ltd.

```
                     )
  {
//--- 根据比例更改指标宽度
   if(ChartGetInteger(0,CHART_SCALE)>3)
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
   else
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
  }
```

另 见

FileWriteString

© 2000-2025, MetaQuotes Ltd.

# FileReadString

函数从文件指标当前位置读取字符串。

```
string  FileReadString(
   int  file_handle,     // 文件句柄
   int  length=-1        // 字符串长度
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回的文件描述符。

*length=-1*

[in] 读取的字符号。

返回值

阅读行（字符串）。

注释

当从二倍文件阅读时，字符串的长度阅读需要指定。从txt文件中不要求阅读字符串长度，字符串从当前位置填满 "\r\n"字节，当从csv文件阅读时，也不要求字符串长度，字符串从当前位置直到最近的定界符或者到文本字符串末尾字节开始阅读。

如果文件使用 FILE_ANSI标记，打开,线路阅读转换到同一码。

示例（执行FileWriteInteger函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 读取数据的参数
input string InpFileName="Trend.bin"; // 文件名称
input string InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 附加变量
      int    str_size;
      string str;
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 找出多少交易品种用于编写时间
         str_size=FileReadInteger(file_handle,INT_VALUE);
         //--- 阅读字符串
         str=FileReadString(file_handle,str_size);
         //--- 打印字符串
         PrintFormat(str);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

字符串类型, 数据转换, FileWriteInteger

© 2000-2025, MetaQuotes Ltd.

# FileReadStruct

从二进制文件中函数读取以参量传递的结构，从文件指针的当前仓位开始。

```
uint  FileReadStruct(
   int           file_handle,        // 文件句柄
   const void&   struct_object,      // 读取的目标结构
   int           size=-1             // 字节中结构大小
   );
```

参量

*file_handle*

[in] 打开二进制文件的文件说明符。

*struct_object*

[out] 结构对象，结构不包括字符串，动态数组 或者 虚拟函数。

*size=-1*

[in] 可以读取字节数量，如果大小未标明或者指示值比结构大小要大，需要使用指定结构的精确大小。

返回值

如果成功，函数返回读取的字节数量，文件指标通过相同的字节数量移动。

示例（执行 FileWriteStruct 函数示例后获得的文件在这里使用）

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//---- 图 Label1
#property indicator_label1  "Candles"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
//--- 接收数据的参数
input string  InpFileName="EURUSD.txt"; // 文件名称
input string  InpDirectoryName="Data";  // 目录名称
//+------------------------------------------------------------------+
//| 存储蜡烛图数据的结构                                             |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // 开盘价
   double            close; // 收盘价
   double            high;  // 最高价
   double            low;   // 最低价
   datetime          date;  // 日期
  };
//--- 指标缓冲区
double        open_buff[];
double        close_buff[];
double        high_buff[];
double        low_buff[];
//--- 全局变量
candlesticks cand_buff[];
int          size=0;
int          ind=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
   int default_size=100;
   ArrayResize(cand_buff,default_size);
//--- 打开文件
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH
      //--- 读取文件数据
      while(!FileIsEnding(file_handle))
        {
         //--- 编写数组数据
         FileReadStruct(file_handle,cand_buff[size]);
         size++;
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 检查数组是否超过
         if(size==default_size)
           {
            //--- 增加数组维数
            default_size+=100;
            ArrayResize(cand_buff,default_size);
           }
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- 指标缓冲区映射
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
//--- 空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   ArraySetAsSeries(time,false);
//--- 还未处理的蜡烛图的循环
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 默认为0
      open_buff[i]=0;
      close_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      //--- 检查任何日期是否仍然存在
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- 如果日期一致，使用文件的值
            if(time[i]==cand_buff[j].date)
              {
               open_buff[i]=cand_buff[j].open;
               close_buff[i]=cand_buff[j].close;
               high_buff[i]=cand_buff[j].high;
               low_buff[i]=cand_buff[j].low;
               //--- 增加计数器
               ind=j+1;
               break;
              }
           }
        }
     }
//--- 返回prev_calculated 值，以便下次调用
   return(rates_total);
  }
```

另 见

结构和类, FileWriteStruct

© 2000-2025, MetaQuotes Ltd.

# FileSeek

通过指定仓位相关字节数，函数移动文件指标的仓位。

```
bool  FileSeek(
   int                  file_handle,      // 文件句柄
   long                 offset,           // 字节
   ENUM_FILE_POSITION   origin            // 参考位置
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件描述符。

*offset*

[in]字节转换 (也许获取负值)。

*origin*

[in]移位起始点。可以是 ENUM_FILE_POSITION值中一个。

返回值

如果成功返回true，否则false。调用GetLastError()函数获得错误信息。

注释

如果执行FileSeek()函数结果导致负值转换 (超过文件的水平边界)，文件指标在文件起始运行时建立。

如果仓位的建立超过文件的右边界 (比文件大)，接下来的录入文件不是从文件末尾开始，而是从仓位开始。不确定值为先前文件末尾和新建仓位编辑。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                Demo_FileSeek.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="file.txt";     // 文件名称
input string InpDirectoryName="Data";    // 目录名称
input int    InpEncodingType=FILE_ANSI; // ANSI=32 或 UNICODE=64
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 指定生成随机数字的变量值
   _RandomSeed=GetTickCount();
//--- 字符串起始点位置的变量
   ulong pos[];
   int   size;
//--- 重置错误值
   ResetLastError();
//--- 打开文件
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- 接收文件中每个字符串的起始位置
      GetStringPositions(file_handle,pos);
      //--- 定义文件中的字符串数量
      size=ArraySize(pos);
      if(!size)
        {
         //--- 如果文件没有字符串，则停止
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- 随机选择一个字符串数
      int ind=MathRand()%size;
      //--- 字符串起始点的转换位置
      if(FileSeek(file_handle,pos[ind],SEEK_SET)==true)
        {
      //--- 阅读和打印ind数字的字符串
         PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_hand
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError());
  }
//+------------------------------------------------------------------------------+
//| 函数定义了文件中每个字符串的起始点并在arr数组放置它                          |
//|                                                                              |
//+------------------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
//--- 默认数组大小
   int def_size=127;
//--- 为数组分配内存
   ArrayResize(arr,def_size);
//--- 字符串计数器
   int i=0;
//--- 如果这不是文件结尾，那么至少有一个字符串
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // 文件为空，退出
//--- 根据编码定义字节转换
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- 通过循环中的字符串
   while(1)
     {
      //--- 阅读字符串
      FileReadString(handle);
      //--- 检查文件结尾
      if(!FileIsEnding(handle))
        {
         //--- 存储下一个字符串的位置
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- 如果超过增加数组大小
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // 文件结尾，退出
     }
//--- 定义数组的实际大小
   ArrayResize(arr,i);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 文件大小

函数返回文件字节大小。

```
ulong  FileSize(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件描述符。

返回值

整型值。

注释

调用GetLastError()函数获得错误信息。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input ulong  InpThresholdSize=20;         // 千字节的文件门槛大小
input string InpBigFolderName="big";      // 大型文件的文件夹
input string InpSmallFolderName="small"; // 小型文件的文件夹
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;       // 存储文件名称的变量
   string   filter="*.csv"; // 搜索文件的过滤器
   ulong    file_size=0;     // 文件的字节大小
   int      size=0;          // 文件数
//--- 打印我们将要使用的文件路径
   PrintFormat("Working in %s\\Files\\ folder",TerminalInfoString(TERMINAL_COMMONDATA_
//--- 接收所有程序端普通文件夹根目录下的搜索程序
   long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);
//--- 检查FileFindFirst()是否成功执行
   if(search_handle!=INVALID_HANDLE)
     {
      //--- 根据大小循环移动文件
      do
        {
         //--- 打开文件
         ResetLastError();
         int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON);
         if(file_handle!=INVALID_HANDLE)
           {
            //--- 接收文件大小
            file_size=FileSize(file_handle);
            //--- 关闭文件
            FileClose(file_handle);
           }
         else
           {
            PrintFormat("Failed to open %s file, Error code = %d",file_name,GetLastErr
            continue;
           }
         //--- 打印文件大小
         PrintFormat("Size of %s file is equal to %d bytes",file_name,file_size);
         //--- 定义移动文件的路径
         string path;
         if(file_size>InpThresholdSize*1024)
            path=InpBigFolderName+"//"+file_name;
         else
            path=InpSmallFolderName+"//"+file_name;
         //--- 移动文件
         ResetLastError();
         if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))
            PrintFormat("%s file is moved",file_name);
         else
            PrintFormat("Error, code = %d",GetLastError());
        }
      while(FileFindNext(search_handle,file_name));
      //--- 关闭搜索程序
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# FileTell

文件返回打开文件指针的当前仓位。

```
ulong  FileTell(
   int  file_handle    // 文件句柄
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件描述符。

返回值

从文件开始，文件说明符的当前仓位字节数。

注释

调用GetLastError()函数获得错误信息。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                Demo_FileTell.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 输入参数
input string InpFileName="file.txt";     // 文件名称
input string InpDirectoryName="Data";    // 目录名称
input int    InpEncodingType=FILE_ANSI; // ANSI=32 或 UNICODE=64
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 指定生成随机数字的变量值
   _RandomSeed=GetTickCount();
//--- 字符串起始点位置的变量
   ulong pos[];
   int   size;
//--- 重置错误值
   ResetLastError();
//--- 打开文件
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- 接收文件中每个字符串的起始位置
      GetStringPositions(file_handle,pos);
      //--- 定义文件中的字符串数量
      size=ArraySize(pos);
      if(!size)
        {
         //--- 如果文件没有字符串，则停止
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- 随机选择一个字符串数
      int ind=MathRand()%size;
      //--- 字符串起始点的转换位置
      FileSeek(file_handle,pos[ind],SEEK_SET);
      //--- 阅读和打印ind数字的字符串
      PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_handle)
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
//+------------------------------------------------------------------+
//| 函数定义了文件中每个字符串的起始点并在arr数组放置它             |
//|                                                                  |
//+------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 默认数组大小
   int def_size=127;
//--- 为数组分配内存
   ArrayResize(arr,def_size);
//--- 字符串计数器
   int i=0;
//--- 如果这不是文件结尾，那么至少有一个字符串
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // 文件为空，退出
//--- 根据编码定义字节转换
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- 通过循环中的字符串
   while(1)
     {
      //--- 阅读字符串
      FileReadString(handle);
      //--- 检查文件结尾
      if(!FileIsEnding(handle))
        {
         //--- 存储下一个字符串的位置
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- 如果超过增加数组大小
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // 文件结尾，退出
     }
//--- 定义数组的实际大小
   ArrayResize(arr,i);
  }
```

© 2000-2025, MetaQuotes Ltd.

# FileWrite

函数预定给CSV文件录入数据，除非等于0，定界符自动插入，在编辑文件中，添加最后字节线"\r\n"。

```
uint  FileWrite(
   int  file_handle,   // 文件句柄
   ...                 // 记录参量的列表
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*...*

[in]参量列表由逗号分开，编辑入文件中，编辑参量数量最高为63。

返回值

编辑字节数。

注释

数量转换成文本输出 (参看Print()函数)。双精度数据以小数点后16位精确度输出，数据以传统或科学模式显示-依据该模式是最紧凑的。浮点型数据以小数点后5位显示，为了以不同精确度输出真实数字或以清晰的指定模式，使用 DoubleToString()。

布尔类型数量以true" 或者 "false"字符串显示，日期时间类型数量以"YYYY.MM.DD HH:MI:SS"显示。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                Demo_FileWrite.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string             InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // 时间帧
input int                InpFastEMAPeriod=12;              // 快速 EMA 周期
input int                InpSlowEMAPeriod=26;              // 慢速 EMA 周期
input int                InpSignalPeriod=9;                // 不同平均周期
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // 价格类型
input datetime           InpDateStart=D'2012.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string             InpFileName="MACD.csv";  // 文件名称
input string             InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // 复制结束日期的数据
   bool     sign_buff[]; // 信号数组 (true - 买入,false - 卖出)
   datetime time_buff[]; // 信号到达时间数组
   int      sign_size=0; // 信号数组大小
   double   macd_buff[]; // 指标值数组
   datetime date_buff[]; // 指标日期数组
   int      macd_size=0; // 指标数组大小
//--- 结束时间是当前时间
   date_finish=TimeCurrent();
//--- 接收 MACD 指标句柄
   ResetLastError();
   int macd_handle=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,InpSlowEMAPeri
   if(macd_handle==INVALID_HANDLE)
     {
      //--- 接收指标句柄失败
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErro
      return;
     }
//--- 循环直至指标计算所有值
   while(BarsCalculated(macd_handle)==-1)
      Sleep(10); // 暂停允许指标计算所有值
//--- 复制某段时间周期的指标值
   ResetLastError();
   if(CopyBuffer(macd_handle,0,InpDateStart,date_finish,macd_buff)==-1)
     {
      PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
      return;
     }
//--- 复制相应的指标值时间
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 释放指标占据的内存
   IndicatorRelease(macd_handle);
//--- 接收缓冲区大小
   macd_size=ArraySize(macd_buff);
//--- 分析数据，保存数组的指标信号
   ArrayResize(sign_buff,macd_size-1);
   ArrayResize(time_buff,macd_size-1);
   for(int i=1;i<macd_size;i++)
     {
      //--- 买入信号
      if(macd_buff[i-1]<0 && macd_buff[i]>=0)
        {
         sign_buff[sign_size]=true;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
      //--- 卖出信号
      if(macd_buff[i-1]>0 && macd_buff[i]<=0)
        {
         sign_buff[sign_size]=false;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
     }
//--- 打开编写指标值的文件 如果文件不在，则自动创建）
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 首先，写下信号的数量
      FileWrite(file_handle,sign_size);
      //--- 写下文件的信号时间和信号值
      for(int i=0;i<sign_size;i++)
         FileWrite(file_handle,time_buff[i],sign_buff[i]);
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

注释，打印，字符串格式化

© 2000-2025, MetaQuotes Ltd.

# FileWriteArray

函数可以编辑除字符串类型外任何类型的数组到BIN文件中（可以是不包含字符串或动态数组的结构数组）。

```
uint  FileWriteArray(
   int           file_handle,          // 文件句柄
   const void&   array[],              // 数组
   int           start=0,              // 数字中启动索引
   int           count=WHOLE_ARRAY     // 单元号
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*array[]*

[out] 记录数组。

*start=0*

[in] 数组的原始索引（第一个记录元素的号码）。

*count=WHOLE_ARRAY*

[in] 记录项目数量（WHOLE_ARRAY 表示记录所有以 start 开头的项目指导数组末尾）。

返回值

记录项目数量。

注释

字符串数组可以记录在TXT文件中，字符串可以自动以 "\r\n"字节结尾，依据文件类型ANSI 或者 UNICODE，字符串可以转换成ANSI编码。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteArray.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 输入参数
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| 存储价格数据的结构                                                 |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // 日期
   double            bid;  // 买入价
   double            ask;  // 卖出价
  };
//--- 全局变量
int    count=0;
int    size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];
//+------------------------------------------------------------------+
//| 专家初始化函数                                                     |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 为数组分配内存
   ArrayResize(arr,size);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家去初始化函数                                                   |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 如果计数，编写其余的计数字符串<n
   WriteData(count);
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                     |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- 保存数组数据
   arr[count].date=TimeCurrent();
   arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- 显示当前数据
   Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[count].ask);
//--- 增加计数器
   count++;
//--- 如果填写数组，向文件写入数据和零
   if(count==size)
     {
      WriteData(size);
      count=0;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   }
//+------------------------------------------------------------------+
//| 向文件写下数组的n元素                                            |
//+------------------------------------------------------------------+
void WriteData(const int n)
  {
//--- 打开文件
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
   if(handle!=INVALID_HANDLE)
     {
      //--- 向文件结尾写下数组数据
      FileSeek(handle,0,SEEK_END);
      FileWriteArray(handle,arr,0,n);
      //--- 关闭文件
      FileClose(handle);
     }
   else
      Print("Failed to open the file, error ",GetLastError());
  }
```

另见

变量，FileSeek

© 2000-2025, MetaQuotes Ltd.

# FileWriteDouble

函数编辑双精度参量到文件中，从文件指针的当前仓位开始。

```
uint  FileWriteDouble(
   int     file_handle,     // 文件句柄
   double  value            // 书写值
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*value*

[in] 双精度类型值。

返回值

如果成功函数返回编辑的字节数量 (在此情况下 sizeof(双精度)=8). 文件指标以相同数量字节转换。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                        Demo_FileWriteDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string             InpSymbolName="EURJPY";           // 货币组
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_M15;       // 时间帧
input int                InpMAPeriod=10;                   // 平滑期
input int                InpMAShift=0;                     // 指标变化
input ENUM_MA_METHOD     InpMAMethod=MODE_SMA;             // 平滑类型
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // 价格类型
input datetime           InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string             InpFileName="MA.csv";     // 文件名称
input string             InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   ma_buff[];
   datetime time_buff[];
   int      size;
//--- 接收 MA 指标句柄
   ResetLastError();
   int ma_handle=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,InpMAMethod,
   if(ma_handle==INVALID_HANDLE)
     {
      //--- 接收指标句柄失败
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErro
      return;
     }
//--- 循环直至指标计算所有值
   while(BarsCalculated(ma_handle)==-1)
      Sleep(20); // 暂停允许指标计算所有值
   PrintFormat("Indicator values starting from %s will be written to the file",TimeToS
//--- 复制指标值
   ResetLastError();
   if(CopyBuffer(ma_handle,0,InpDateStart,date_finish,ma_buff)==-1)
     {
      PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError
      return;
     }
//--- 复制相应的柱形到达时间
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- 接收缓冲区大小
   size=ArraySize(ma_buff);
//--- 释放指标占据的内存
   IndicatorRelease(ma_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 打开编写指标值的文件 如果文件不在，则自动创建）
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 首先，写下数据样本的大小
      FileWriteDouble(file_handle,(double)size);
      //--- 写下文件的指标时间和值
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteDouble(file_handle,ma_buff[i]);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

真实型（双精度型，浮点型）

© 2000-2025, MetaQuotes Ltd.

# FileWriteFloat

函数编辑浮点参量的值到二次文件中，从文件指标当前仓位开始。

```
uint  FileWriteFloat(
   int    file_handle,     // 文件句柄
   float  value            // 被写入的值
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符

*value*

[in]浮点类型值。

返回值

如果成功函数返回编辑的字节数量(在此情况下 sizeof(浮点类型)=4). 文件指标以相同数量字节转换。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string          InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;       // 时间帧
input datetime        InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string          InpFileName="Close.bin"; // 文件名称
input string          InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- 重置错误值
   ResetLastError();
//--- 复制每个柱的收盘价
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1
     {
      PrintFormat("Failed to copy close price values. Error code = %d",GetLastError())
      return;
     }
//--- 复制每个柱形的时间
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy the time values. Error code = %d",GetLastError());
      return;
     }
//--- 接收缓冲区大小
   size=ArraySize(close_buff);
//--- 打开编写指标值的文件（如果文件不在，则自动创建）
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 写下文件的收盘价时间和值
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteFloat(file_handle,(float)close_buff[i]);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另见

真实型（双精度型，浮点型）, FileWriteDouble

© 2000-2025, MetaQuotes Ltd.

# FileWriteInteger

函数编辑整型参量值到二次文件中，从文件指标当前仓位开始。

```
uint  FileWriteInteger(
   int  file_handle,          // 文件句柄
   int  value,                // 被编写值
   int  size=INT_VALUE        // 字节大小
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符

*value*

[in] 整数值。

*size=INT_VALUE*

[in] 字节数量 (最多包括4字节)可以编辑。提供类似常量：CHAR_VALUE=1, SHORT_VALUE=2 和 INT_VALUE=4,因此函数可以编辑图表型，双字符型，短型，长短型，整型或者长整型的整数值。

返回值

如果成功函数返回字节编辑数量，文件指标以相同数量字节转换。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                      Demo_FileWriteInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string              InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES     InpSymbolPeriod=PERIOD_H1;        // 时间帧
input datetime            InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string              InpFileName="Trend.bin"; // 文件名称
input string              InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- 重置错误值
   ResetLastError();
//--- 复制每个柱的收盘价
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1
     {
      PrintFormat("Failed to copy the values of close prices. Error code = %d",GetLast
      return;
     }
//--- 复制每个柱形的时间
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- 接收缓冲区大小
   size=ArraySize(close_buff);
//--- 打开编写值的文件 如果文件不在，则自动创建）
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //---
      int   up_down=0; // 趋势标识
      int   arr_size;  // arr 数组大小
      uchar arr[];     // uchar 类型数组
      //--- 写下文件的时间值
      for(int i=0;i<size-1;i++)
        {
         //--- 比较当前柱和下一柱的收盘价
         if(close_buff[i]<=close_buff[i+1])
           {
            if(up_down!=1)
              {
```

© 2000-2025, MetaQuotes Ltd.

```
                //--- 使用FileWriteInteger写下文件的日期值
                StringToCharArray(TimeToString(time_buff[i]),arr);
                arr_size=ArraySize(arr);
                //--- 首先，写下数组中的交易品种数量
                FileWriteInteger(file_handle,arr_size,INT_VALUE);
                //--- 写下交易品种
                for(int j=0;j<arr_size;j++)
                   FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
                //--- 改变趋势标识
                up_down=1;
               }
            }
          else
            {
             if(up_down!=-1)
               {
                //--- 使用FileWriteInteger写下文件的日期值
                StringToCharArray(TimeToString(time_buff[i]),arr);
                arr_size=ArraySize(arr);
                //--- 首先，写下数组中的交易品种数量
                FileWriteInteger(file_handle,arr_size,INT_VALUE);
                //--- 写下交易品种
                for(int j=0;j<arr_size;j++)
                   FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
                //--- 改变趋势标识
                up_down=-1;
               }
            }
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

整数到字符串, 字符串到整数, 整数类型

© 2000-2025, MetaQuotes Ltd.

# FileWriteLong

函数编辑长型参量值到二进制文件中，从文件指标当前仓位开始。

```
uint  FileWriteLong(
   int    file_handle,      // 文件句柄
   long   value             // 被写入的值
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*value*

[in]长整型值。

返回值

如果成功函数返回编辑字节数量(在此情况下 sizeof(长型)=8).文件指标以相同数量字节转变。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string          InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // 时间帧
input datetime        InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string          InpFileName="Volume.bin"; // 文件名称
input string          InpDirectoryName="Data";  // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   long     volume_buff[];
   datetime time_buff[];
   int      size;
//--- 重置错误值
   ResetLastError();
//--- 复制每个柱的跳动量
   if(CopyTickVolume(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,volume_bu
     {
      PrintFormat("Failed to copy values of the tick volume. Error code = %d",GetLastE
      return;
     }
//--- 复制每个柱形的时间
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- 接收缓冲区大小
   size=ArraySize(volume_buff);
//--- 打开编写指标值的文件（如果文件不在，则自动创建）
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 首先，写下数据样本的大小
      FileWriteLong(file_handle,(long)size);
      //--- 写下文件的时间和交易量值
      for(int i=0;i<size;i++)
        {
         FileWriteLong(file_handle,(long)time_buff[i]);
         FileWriteLong(file_handle,volume_buff[i]);
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

整数类型, FileWriteInteger

© 2000-2025, MetaQuotes Ltd.

# FileWriteString

该函数编写字符串类型参数的值到文件指针当前位置开始的BIN,CSV或TXT文件。当写下CSV或TXT文件时：如果无之前字符 '\r' (CR)的 '\n' (LF)字符串中有一个交易品种，那么'\n'之前会添加丢失的 '\r'。

```
uint  FileWriteString(
   int           file_handle,     // 文件句柄
   const string  text_string,     // 要编写的字符串
   int           length=-1        // 交易品种数量
   );
```

参量

*file_handle*

[in] 通过 FileOpen()返回文件说明符。

*text_string*

[in] 字符串。

*length=-1*

[in] 想要编辑的字节数量，选项需要从字符串编辑到BIN文件中，如果大小未指定，全部末尾没有0的字符串都能编辑，如果指定大小小于字符串长度，只能编辑一部分的非零字符串。如果指定大小大于字符串的长度，字符串会占用0字符。

对于CSV 和 TXT类型的文件，该参数可以忽略并且字符串完全编写。

返回值

如果成功，函数返回字节编辑数量，文件指标以相同字节数量转换。

注释

当通过FILE_UNICODE 标记(或者没有 FILE_ANSI标签)打开编辑文件，编辑的字节数量是字符串字节编辑的两倍大。当记录文件以FILE_ANSI标签打开，字节编辑数量会与字符串字节编辑数量相一致。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                        Demo_FileWriteString.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string             InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // 时间帧
input int                InpMAPeriod=14;                   // MA周期
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // 价格类型
input datetime           InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string             InpFileName="RSI.csv";    // 文件名称
input string             InpDirectoryName="Data"; // 目录名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // 复制结束日期的数据
   double   rsi_buff[];  // 指标值数组
   datetime date_buff[]; // 指标日期数组
   int      rsi_size=0;  // 指标数组大小
//--- 结束时间是当前时间
   date_finish=TimeCurrent();
//--- 接收 RSI 指标句柄
   ResetLastError();
   int rsi_handle=iRSI(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- 接收指标句柄失败
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErro
      return;
     }
//--- 循环直至指标计算所有值
   while(BarsCalculated(rsi_handle)==-1)
      Sleep(10); // 暂停允许指标计算所有值
//--- 复制某段时间周期的指标值
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,date_finish,rsi_buff)==-1)
     {
      PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
      return;
     }
//--- 复制相应的指标值时间
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- 释放指标占据的内存
   IndicatorRelease(rsi_handle);
//--- 接收缓冲区大小
   rsi_size=ArraySize(rsi_buff);
//--- 打开编写指标值的文件 如果文件不在，则自动创建）
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- 准备附加变量
      string str="";
      bool   is_formed=false;
      //--- 写下形成超买和超卖区域的日期
      for(int i=0;i<rsi_size;i++)
        {
         //--- 检查指标值
         if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
           {
            //--- 如果该值是该区域的第一个值
            if(!is_formed)
              {
               //--- 添加值和日期
               str=(string)rsi_buff[i]+"\t"+(string)date_buff[i];
               is_formed=true;
              }
            else
               str+="\t"+(string)rsi_buff[i]+"\t"+(string)date_buff[i];
            //--- 移动到下个循环迭代
            continue;
           }
         //--- 检查标识
         if(is_formed)
           {
            //--- 字符串形成，将其写入文件
            FileWriteString(file_handle,str+"\r\n");
            is_formed=false;
           }
        }
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

字符串类型，字符串格式化

© 2000-2025, MetaQuotes Ltd.

# FileWriteStruct

通过传递参量函数编辑二次结构文件目录，从文件指标的当前仓位开始。

```
uint  FileWriteStruct(
   int           file_handle,       // 文件句柄
   const void&   struct_object,     // 连接一个物件
   int           size=-1            // 写入字节大小
   );
```

参量

*file_handle*

[in] 通过FileOpen()返回文件说明符

*struct_object*

[in]参考结构对象，结构不应该包含字符串，动态数组或者虚拟函数。

*size=-1*

[in]想要记录的自己数量，如果大小不够指定或者指定字节数量大于结构大小，编辑全部结构。

返回值

如果成功，函数返回编辑的自己数量，文件指标以相同数量字节转换。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWiteStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 启动脚本时显示输入参数的窗口
#property script_show_inputs
//--- 接收程序端数据的参数
input string          InpSymbolName="EURUSD";           // 货币组
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // 时间帧
input datetime        InpDateStart=D'2013.01.01 00:00'; // 复制起始日期的数据
//--- 编写文件数据的参数
input string          InpFileName="EURUSD.txt";         // 文件名称
input string          InpDirectoryName="Data";          // 目录名称
//+------------------------------------------------------------------+
//| 存储蜡烛图数据的结构                                             |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // 开盘价
   double            close; // 收盘价
   double            high;  // 最高价
   double            low;   // 最低价
   datetime          date;  // 日期
  };
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime     date_finish=TimeCurrent();
   int          size;
   datetime     time_buff[];
   double       open_buff[];
   double       close_buff[];
   double       high_buff[];
   double       low_buff[];
   candlesticks cand_buff[];
//--- 重置错误值
   ResetLastError();
//--- 接收范围内柱形到达时间
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- 接收范围内柱形的最高价
   if(CopyHigh(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,high_buff)==-1)
     {
      PrintFormat("Failed to copy values of high prices. Error code = %d",GetLastErro
      return;
     }
//--- 接收范围内柱形的最低价
   if(CopyLow(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,low_buff)==-1)
     {
      PrintFormat("Failed to copy values of low prices. Error code = %d",GetLastError
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 接收范围内柱形的开盘价
   if(CopyOpen(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,open_buff)==-1)
     {
      PrintFormat("Failed to copy values of open prices. Error code = %d",GetLastError
      return;
     }
//--- 接收范围内柱形的收盘价
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1
     {
      PrintFormat("Failed to copy values of close prices. Error code = %d",GetLastErro
      return;
     }
//--- 定义数组维数
   size=ArraySize(time_buff);
//--- 在结构数组保存所有数据
   ArrayResize(cand_buff,size);
   for(int i=0;i<size;i++)
     {
      cand_buff[i].open=open_buff[i];
      cand_buff[i].close=close_buff[i];
      cand_buff[i].high=high_buff[i];
      cand_buff[i].low=low_buff[i];
      cand_buff[i].date=time_buff[i];
     }

//--- 打开用于编写文件结构数组的文件(如果文件不在，则自动创建)
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH
      //--- 准备字节数量的计数器
      uint counter=0;
      //--- 写下循环数组值
      for(int i=0;i<size;i++)
         counter+=FileWriteStruct(file_handle,cand_buff[i]);
      PrintFormat("%d bytes of information is written to %s file",InpFileName,counter)
      PrintFormat("Total number of bytes: %d * %d * %d = %d, %s",size,5,8,size*5*8,siz
      //--- 关闭文件
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

另见

结构和类

© 2000-2025, MetaQuotes Ltd.

# FileLoad

指定二进制文件的所有数据读入数值类型或简单结构的传递数组。该函数允许您快速的将已知类型的数据读入相应的数组。

```
long  FileLoad(
   const string  file_name,          // 文件名
   void&         buffer[],           // 数值类型或简单结构的数组
   int           common_flag=0       // 在<data_folder>\MQL5\Files\ by default中搜索的文
   );
```

参数

*file_name*

[in] 阅读数据的文件名称。

*buffer*

[out] 数值类型或 简单结构数组。

*common_flag=0*

[in] 指示操作模式的文件标识。如果没有指定参数，文件将在子文件夹MQL5\Files被搜索(或测试时在<testing_agent_directory>\MQL5\Files)。

返回值

读取的元素数量或错误时为 -1。

注意

FileLoad()从文件读取数组大小几倍的字节数。假设文件大小为10字节，函数将数据读入双精度类型数组 (sizeof(double)=8)。在这种情况下，函数只能读入8个字节，在文件结尾剩下的2个字节将被删除，函数FileLoad()将返回1 (1个元素阅读)。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_FileLoad.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- 输入参数
input int      bars_to_save=10; // Number of bars
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnStart()
  {
   string  filename=_Symbol+"_rates.bin";
   MqlRates rates[];
//---
   int copied=CopyRates(_Symbol,_Period,0,bars_to_save,rates);
   if(copied!=-1)
     {
      PrintFormat(" CopyRates(%s) copied %d bars",_Symbol,copied);
      //---  报价写入文件
      if(!FileSave(filename,rates,FILE_COMMON))
         PrintFormat("FileSave() failed, error=%d",GetLastError());
     }
   else
      PrintFormat("Failed CopyRates(%s), error=",_Symbol,GetLastError());
//--- 现在阅读这些返回到文件的报价
   ArrayFree(rates);
   long count=FileLoad(filename,rates,FILE_COMMON);
   if(count!=-1)
     {
      Print("Time\tOpen\tHigh\tLow\tClose\tTick Voulme\tSpread\tReal Volume");
      for(int i=0;i<count;i++)
        {
         PrintFormat("%s\t%G\t%G\t%G\t%G\t%I64u\t%d\t%I64u",
                     TimeToString(rates[i].time,TIME_DATE|TIME_SECONDS),
                     rates[i].open,rates[i].high,rates[i].low,rates[i].close,
                     rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
     }
  }
```

另见

Structures and Classes, FileReadArray, FileReadStruct, FileSave

© 2000-2025, MetaQuotes Ltd.

# FileSave

将传递数组所有元素写入二进制文件作为参数。该函数允许您快速编写数值类型或简单结构的数组为一个字符串。

```
bool  FileSave(
   const string  file_name,          // 文件名
   void&         buffer[],           // 数值类型或简单结构的数组
   int           common_flag=0       // 文件标识，默认文件写入<data_folder>\MQL5\Files\
   );
```

参数

*file_name*

[in] 将被写入数据数组的文件名称。

*buffer*

[in] 数值类型或 简单结构数组。

*common_flag=0*

[in] 指示操作模式的文件标识。如果没有指定参数，文件将被写入子文件夹MQL5\Files(或测试时在<testing_agent_directory>\MQL5\Files)。

返回值

如果失败返回false。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_FileSave.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- 输入参数
input int      ticks_to_save=1000; // Number of ticks
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string  filename=_Symbol+"_ticks.bin";
   MqlTick ticks[];
u//---
   int copied=CopyTicks(_Symbol,ticks,COPY_TICKS_ALL,0,ticks_to_save);
   if(copied!=-1)
     {
      PrintFormat(" CopyTicks(%s) copied %d ticks",_Symbol,copied);
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果报价历史被同步，错误代码等于零
      if(!GetLastError()==0)
         PrintFormat("%s: Ticks are not synchronized, error=%d",_Symbol,copied,_LastEr
      //---  报价写入文件
      if(!FileSave(filename,ticks,FILE_COMMON))
         PrintFormat("FileSave() failed, error=%d",GetLastError());
     }
   else
      PrintFormat("Failed CopyTicks(%s), Error=",_Symbol,GetLastError());
//--- 现在将这些报价读入文件
   ArrayFree(ticks);
   long count=FileLoad(filename,ticks,FILE_COMMON);
   if(count!=-1)
     {
      Print("Time\tBid\tAsk\tLast\tVolume\tms\tflags");
      for(int i=0;i<count;i++)
        {
         PrintFormat("%s.%03I64u:\t%G\t%G\t%G\t%I64u\t0x%04x",
         TimeToString(ticks[i].time,TIME_DATE|TIME_SECONDS),ticks[i].time_msc%1000,
         ticks[i].bid,ticks[i].ask,ticks[i].last,ticks[i].volume,ticks[i].flags);
        }
     }
  }
```

另见

Structures and Classes, FileWriteArray, FileWriteStruct, FileLoad, FileWrite

© 2000-2025, MetaQuotes Ltd.

# FolderCreate

函数在文件目录下新建目录（依据 common_flag的值）。

```
bool  FolderCreate(
   string  folder_name,        // 带有新文件夹名的字符串
   int     common_flag=0       // 范围
   );
```

参量

*folder_name*

[in] Name of the directory to be created. Contains the relative path to the folder.

*common_flag=0*

[in] 标记 决定目录位置，如果common_flag=FILE_COMMON,那么目录在所有客户端的共享目录中 \Terminal\Common\Files，否则，目录会在本地文件夹中（MQL5\Files 或者 MQL5\Tester\Files）。

返回值

如果成功返回true，否则-false。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实 施文件操作的文件意味着，不能在文件沙箱外。

示例：

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- description
#property description "The script shows FolderCreate() application sample."
#property description "The external parameter defines the directory for creating folde
#property description "The folder structure is created after executing the script"

//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- the input parameter defines the folder, in which the script works
input bool     common_folder=false; // common folder for all terminals
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- folder to be created in MQL5\Files
   string root_folder="Folder_A";
   if(CreateFolder(root_folder,common_folder))
     {
      //--- create the Child_Folder_B1 sub-folder in it
```

© 2000-2025, MetaQuotes Ltd.

```
      string folder_B1="Child_Folder_B1";
      string path=root_folder+"\\"+folder_B1;          // create the folder name consi
      if(CreateFolder(path,common_folder))
        {
         //--- create 3 more sub-directories in this folder
         string folder_C11="Child_Folder_C11";
         string child_path=root_folder+"\\"+folder_C11;// create the folder name consi
         CreateFolder(child_path,common_folder);
         //--- second sub-directory
         string folder_C12="Child_Folder_C12";
         child_path=root_folder+"\\"+folder_C12;
         CreateFolder(child_path,common_folder);

         //--- third sub-directory
         string folder_C13="Child_Folder_C13";
         child_path=root_folder+"\\"+folder_C13;
         CreateFolder(child_path,common_folder);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| Try creating a folder and display a message about that           |
//+------------------------------------------------------------------+
bool CreateFolder(string folder_path,bool common_flag)
  {
   int flag=common_flag?FILE_COMMON:0;
   string working_folder;
//--- define the full path depending on the common_flag parameter
   if(common_flag)
      working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL5\\Files";
   else
      working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- debugging message
   PrintFormat("folder_path=%s",folder_path);
//--- attempt to create a folder relative to the MQL5\Files path
   if(FolderCreate(folder_path,flag))
     {
      //--- display the full path for the created folder
      PrintFormat("Created the folder %s",working_folder+"\\"+folder_path);
      //--- reset the error code
      ResetLastError();
      //--- successful execution
      return true;
     }
   else
      PrintFormat("Failed to create the folder %s. Error code %d",working_folder+folde
//--- execution failed
   return false;
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

另 见

FileOpen(), FolderClean(), FileCopy()

© 2000-2025, MetaQuotes Ltd.

# FolderDelete

函数删除指定目录，如果文件夹不是空的，就不能被删除。

```
bool  FolderDelete(
   string  folder_name,        // 带有要删除的文件夹名的字符串
   int     common_flag=0       // 范围
   );
```

参量

*folder_name*

[in] 想要删除的目录名称，包括文件夹的全部路径。

*common_flag=0*

[in] Flag 决定目录位置，如果common_flag=FILE_COMMON,那么目录在所有客户端的共享目录中 \Terminal\Common\Files，否则，目录会在本地文件夹中(MQL5\Files 或者MQL5\tester\Files)。

返回值

如果成功返回true，否则false。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

如果目录至少包括一个文件和/或子目录，目录就不能删除，必须先清理。使用 FolderClean() 清理文件夹中所有文件或者子目录。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                              Demo_FolderDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 描述
#property description "The script shows a sample use of FolderDelete()."
#property description "First two folders are created; one of them is empty, the second
#property description "When trying to delete a non-empty folder, an error is returned

//--- 启动脚本时显示输入参数对话框
#property script_show_inputs
//--- 输入参数
input string   firstFolder="empty";     // 空文件夹
input string   secondFolder="nonempty";// 将会创建文件的文件夹
string filename="delete_me.txt";        // 将在secondFolder文件夹创建的文件名称
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 在这里写下文件句柄
   int handle;
//--- 找出我们正在工作的文件夹
   string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- 调试信息
   PrintFormat("working_folder=%s",working_folder);
//--- 试图创建MQL5\Files路径相关的空文件夹
   if(FolderCreate(firstFolder,0)) // 0 意味着我们在程序端的本地文件夹工作
     {
      //--- 向已创建的文件夹输入全路径
      PrintFormat("Folder %s has been created",working_folder+"\\"+firstFolder);
      //--- 重置错误代码
      ResetLastError();
     }
   else
      PrintFormat("Failed to create folder %s. Error code %d",working_folder+"\\"+firs

//--- 现在使用FileOpen()函数创建一个非空文件夹
   string filepath=secondFolder+"\\"+filename;  // 形成我们想要打开写在不存在文件夹中的路径
   handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // 这种情况下FILE_WRITE标识是必须的，请
   if(handle!=INVALID_HANDLE)
      PrintFormat("File %s has been opened for reading",working_folder+"\\"+filepath);
   else
      PrintFormat("Failed to create file %s in folder %s. Error code=",filename,second

   Comment(StringFormat("Prepare to delete folders %s and %s", firstFolder, secondFold
//--- 短暂停顿5秒来阅读图表中的信息
   Sleep(5000); // Sleep() 不能用在指标中!

//--- 显示询问用户的对话框
   int choice=MessageBox(StringFormat("Do you want to delete folders %s and %s?", firs
                         "Deleting folders",
                         MB_YESNO|MB_ICONQUESTION); //  两个按钮 - "Yes" 和 "No"

//--- 根据选定的变量运行一个行为
   if(choice==IDYES)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 从图表删除评论
      Comment("");
      //--- 添加信息到 "专家"日志
      PrintFormat("Trying to delete folders %s and %s",firstFolder, secondFolder);
      ResetLastError();
      //--- 删除空文件夹
      if(FolderDelete(firstFolder))
         //--- 由于文件夹为空，应该会出现以下信息
         PrintFormat("Folder %s has been successfully deleted",firstFolder);
      else
         PrintFormat("Failed to delete folder %s. Error code=%d", firstFolder, GetLast

      ResetLastError();
      //--- 删除包含文件的文件夹
      if(FolderDelete(secondFolder))
         PrintFormat("Folder %s has been successfully deleted", secondFolder);
      else
         //--- 由于文件夹包含文件，应该会出现下列信息
         PrintFormat("Failed to delete folder %s. Error code=%d", secondFolder, GetLas
     }
   else
      Print("Deletion canceled");
//---
  }
```

另 见

FileOpen(), FolderClean(), FileMove()

© 2000-2025, MetaQuotes Ltd.

# FolderClean

函数删除指定文件夹中所有文件。

```
bool  FolderClean(
   string  folder_name,        // 带有已删除的文件夹名的字符串
   int     common_flag=0       // 范围
   );
```

参量

*folder_name*

[in] 想要删除的所有文件的目录名称，包括文件夹的全部路径。

*common_flag=0*

[in] 标记 决定目录位置，如果common_flag=FILE_COMMON,那么目录在所有客户端的共享目录中 \Terminal\Common\Files，否则，目录会在本地文件夹中(MQL5\Files 或者MQL5\Tester\Files)。

返回值

如果成功返回true，否则false。

注释

出于安全考虑，工作文件必须严格由MQL5语言管理。使用MQL5实施文件操作的文件意味着，不能在文件沙箱外。

该函数慎重使用，所有文件和子目录会被彻底删除。

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                Demo_FolderClean.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 描述
#property description "The script shows a sample use of FolderClean()."
#property description "First, files are created in the specified folder using the File
#property description "Then, before the files are deleted, a warning is shown using Me

//--- 启动脚本时显示输入参数对话框
#property script_show_inputs
//--- 输入参数
input string   foldername="demo_folder";  // 在MQL5/Files/创建一个文件夹
input int      files=5;                   // 要创建和删除的文件数量
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string name="testfile";
//--- 首先在程序端数据文件夹打开或创建文件
   for(int N=0;N<files;N++)
     {
      //--- 'demo_folder\testfileN.txt'格式的文件名称
      string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
      //--- 打开用于编写的标识文件，在这种情况下将会自动创建'demo_folder'
      int handle=FileOpen(filemane,FILE_WRITE);
      //--- 找出FileOpen()函数是否成功
      if(handle==INVALID_HANDLE)
        {
         PrintFormat("Failed to create file %s. Error code",filemane,GetLastError());
         ResetLastError();
        }
      else
        {
         PrintFormat("File %s has been successfully opened",filemane);
         //--- 打开的文件不再需要，所以请关闭它
         FileClose(handle);
        }
     }

//--- 检查文件夹中的文件数量
   int k=FilesInFolder(foldername+"\\*.*",0);
   PrintFormat("Totally the folder %s contains %d files",foldername,k);

//--- 显示询问用户的对话框
   int choice=MessageBox(StringFormat("You are going to delete %d files from folder %s
                         "Deleting files from the folder",
                         MB_YESNO|MB_ICONQUESTION); //  两个按钮 - "Yes" 和 "No"
   ResetLastError();

//--- 根据选定的变量运行一个行为
   if(choice==IDYES)
     {
      //--- 开始删除文件
      PrintFormat("Trying to delete all files from folder %s",foldername);
      if(FolderClean(foldername,0))
```

© 2000-2025, MetaQuotes Ltd.

```
         PrintFormat("Files have been successfully deleted, %d files left in folder %s
                     foldername,
                     FilesInFolder(foldername+"\\*.*",0));
      else
         PrintFormat("Failed to delete files from folder %s. Error code %d",foldername
     }
   else
      PrintFormat("Deletion canceled");
//---
  }
//+------------------------------------------------------------------+
//| 在指定文件夹返回文件数量                                         |
//+------------------------------------------------------------------+
int FilesInFolder(string path,int flag)
  {
   int count=0;
   long handle;
   string filename;
//---
   handle=FileFindFirst(path,filename,flag);
//--- 如果至少一个文件被找到，请搜索更多的文件
   if(handle!=INVALID_HANDLE)
     {
      //--- 显示文件名称
      PrintFormat("File %s found",filename);
      //--- 增加找到的文件/文件夹的计数器
      count++;
      //--- 在所有文件/文件夹中开始搜索
      while(FileFindNext(handle,filename))
        {
         PrintFormat("File %s found",filename);
         count++;
        }
      //--- 完成时不要忘记关闭搜索句柄
      FileFindClose(handle);
     }
   else // 获得句柄失败
     {
      PrintFormat("Files search in folder %s failed",path);
     }
//--- 返回结果
   return count;
  }
```

另 见

FileFindFirst, FileFindNext, FileFindClose

© 2000-2025, MetaQuotes Ltd.

# 自定义指标

这是一组用来创建自定义指标的函数，这些函数不能用于编辑EA交易和脚本。

| 函数 | 功能 |
| --- | --- |
| SetIndexBuffer | 将指定指标缓冲区和一维双精度动态数组绑定一起 |
| IndicatorSetDouble | 设置双精度型指标属性值 |
| IndicatorSetInteger | 设置整型型指标属性值 |
| IndicatorSetString | 设置字符串指标属性值 |
| PlotIndexSetDouble | 设置双精度型指标行属性值 |
| PlotIndexSetInteger | 设置整型指标行属性值 |
| PlotIndexSetString | 设置字符串指标行属性值 |
| PlotIndexGetInteger | 返回整数型指标行属性值 |

指标属性可以使用编译程序指令或使用函数来设定。若要更好的理解这点，建议您学习指标类型示例。

自定义指标中所有必要的运算都在预定义的OnCalculate()函数中，如果短时间使用OnCalculate()函数调用，如

```
int OnCalculate (const int rates_total, const int prev_calculated, const int begin, co
```

而后*rates_total*参量包括price[]数组中的全部元素数量值，以输入参量传递到计算指标值中。

参量 *prev_calculated* 是先前调用的OnCalculate() 计算结果，允许为计算指标值组织一种节约算法，例如，如果当前值是rates_total = 1000, prev_calculated = 999, 足够为每个指标缓冲区计算值。

如果是关于输入数组价格的信息是不能利用的，它会为每个指标缓冲区计算1000个必要值。第一次调用OnCalculate() value prev_calculated = 0，如果price[]数组在某种程度上改变，case prev_calculated 会等于0。

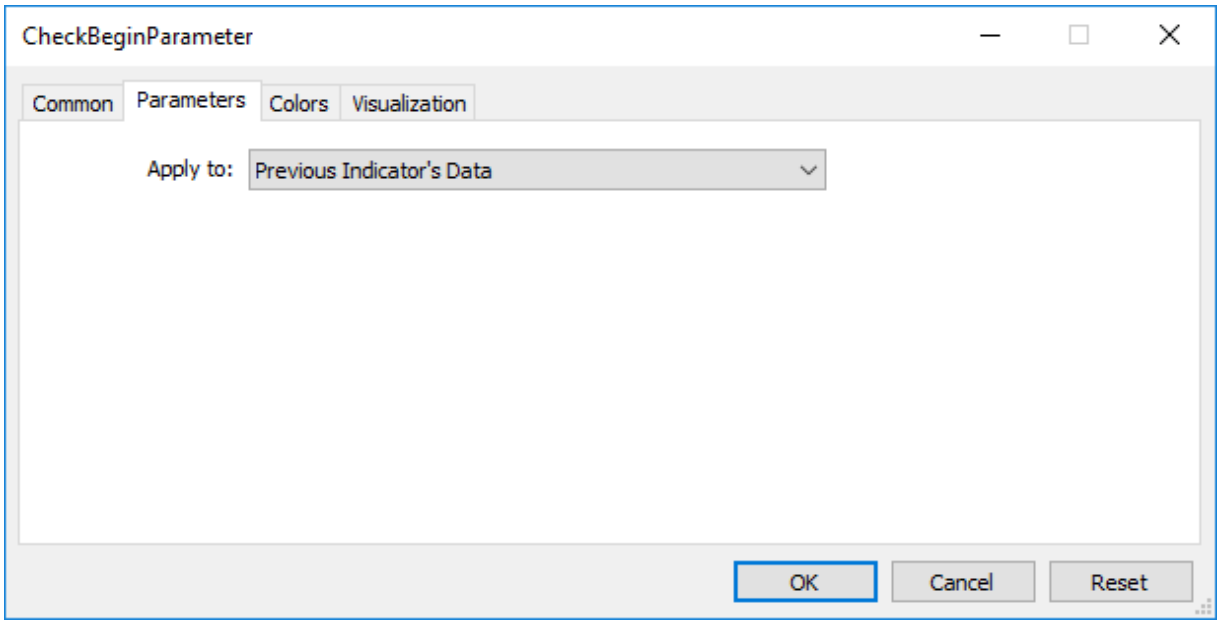
*begin*参量表示价格数组原始值的数量，不包括计算数据。例如，如果Accelerator Oscillator 值 (开头37个值没有计算)以输入参量使用，那么begin = 37。例如，列举一个简单指标：

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 绘图标签1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标缓冲区
double         Label1Buffer[];
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标初始化函数                                                     |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                                     |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])

  {
//---
   Print("begin = ",begin,"  prev_calculated = ",prev_calculated,"  rates_total = ",ra
//--- 为下一次调用返回prev_calculated值
   return(rates_total);
  }
```

从"导航器"界面离开到加速震荡指标窗口，我们认为该计算一定以先前指标的值为基础。




结果是，第一次调用prev_calculated的OnCalculate()值等于0，之后调用等于rates_total值（直到增加价格图表的柱的数量）。

© 2000-2025, MetaQuotes Ltd.

开始参量的值直接等于原始字节数量，加速指标的值不能根据指标逻辑值来计算。如果浏览自定义指标加速器.mq5的源代码，可以看到 OnInit() 函数线：

```
//--- 从被画的指数中设置第一柱
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,37);
```

使用函数PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, empty_first_values)，可以建立自定义指标的0指标数组中非存在第一值的编号，这样，我们不需要接收计算(empty_first_values)。 因此，可以机械使用：

1. 建立指标的原始值数字，不能用了计算另外自定义指标；
2. 当调用另一自定义指标时，获得关于第一忽略值的数量信息，不需要进入计算逻辑值。

© 2000-2025, MetaQuotes Ltd.

# 指标类型示例

MetaTrader 5 客户端包含可以使用相应函数用在MQL5程序的38种技术指标。但是MQL5语言的主要优势在于创建自定义指标的能力，然后将其用在EA交易或简单地应用在价格图表，用于技术分析。

指标的整个设定可以源自于几种基本的 绘图类型，主要为标图。标图是指展示数据的方式，可以用于指标计算，存储和提供请求。共有七种这类的基本标图类型：

1. 线型
2. 节 (段)
3. 直方图
4. 箭头 (符号)
5. 禁区 (填充的通道)
6. 柱形图
7. 日式蜡烛图

每个标图都需要一至五个双精度类型的数组，指标值可以存在这里。为方便起见，这些数组与指标缓冲区相联。一个指标缓冲区的数量必须使用编译程序指令提前声明，例如：

```
#property indicator_buffers 3 // 缓冲区数量
#property indicator_plots   2 // 标图数量
```

指标缓冲区的数量通常大于或等于指标标图的数量。

由于每个基本标图类型都可以有颜色变量或结构规范，所以MQL5标图类型的实际数量是18：

| 标图 | 描述 | 价值缓冲区 | 颜色缓冲区 |
|---|---|---|---|
| DRAW_NONE | 不直观显示在图表中，但是在数据窗口可以看到相应缓冲区的值 | 1 | - |
| DRAW_LINE | 在相应缓冲区的值上标绘线型 (缓冲区的空值不符合) | 1 | - |
| DRAW_SECTION | 绘制为相应缓冲区值之间的 | 1 | - |

© 2000-2025, MetaQuotes Ltd.

| 标图 | 描述 | 价值缓冲区 | 颜色缓冲区 |
| --- | --- | --- | --- |
| | 线段（通常有很多空值） | | |
| DRAW_HISTOGRAM | 绘制为从零线到相应缓冲区值的直方图（可能有很多空值） | 1 | - |
| DRAW_HISTOGRAM2 | 绘制为基于两个指标缓冲区的直方图（可能有空值） | 2 | - |
| DRAW_ARROW | 绘制为符号（可能有空值） | 1 | - |
| DRAW_ZIGZAG | 类似DRAW_SECTION样式，但又不一样，它可以在柱上绘制垂直段 | 2 | - |
| DRAW_FILLING | 颜色填充在两条线之间。相应缓冲区的2个值显示在数据窗口 | 2 | - |
| DRAW_BARS | 绘制为柱形。相应缓冲区的4个值显示 | 4 | - |

© 2000-2025, MetaQuotes Ltd.

| 标图 | 描述 | 价值缓冲区 | 颜色缓冲区 |
| --- | --- | --- | --- |
| | 在数据窗口 | | |
| DRAW_CANDLES | 绘制为日式蜡烛图。相应缓冲区的4个值显示在数据窗口 | 4 | - |
| DRAW_COLOR_LINE | 您可以在不同柱上替换颜色或随时改变其颜色的线 | 1 | 1 |
| DRAW_COLOR_SECTION | 类似DRAW_SECTION样式，但每段的颜色可以单独设置；颜色也可以动态设置 | 1 | 1 |
| DRAW_COLOR_HISTOGRAM | 类似DRAW_HISTOGRAM样式，但每一条都可能有不同的颜色，您可以设置动态颜色 | 1 | 1 |
| DRAW_COLOR_HISTOGRAM2 | 类似DRAW_HISTOGRAM2样式，但每 | 2 | 1 |

© 2000-2025, MetaQuotes Ltd.

| 标图 | 描述 | 价值缓冲区 | 颜色缓冲区 |
| --- | --- | --- | --- |
| | 一条都可能有不同的颜色，您可以设置动态颜色 | | |
| DRAW_COLOR_ARROW | 类似 DRAW_ARROW 样式，但每个符号都可能有它的颜色。颜色可以动态改变 | 1 | 1 |
| DRAW_COLOR_ZIGZAG | DRAW_ZIGZAG 样式具有节段单独颜色和动态颜色改变的选项 | 2 | 1 |
| DRAW_COLOR_BARS | DRAW_BARS 样式具有柱形单独颜色和动态颜色改变的选项 | 4 | 1 |
| DRAW_COLOR_CANDLES | DRAW_CANDLES 样式具有蜡烛图单独颜色和动态颜色改变的选项 | 4 | 1 |

© 2000-2025, MetaQuotes Ltd.

## 指标缓冲区和数组之间的不同性

每个指标中，在其全局水平，您都应该声明一个或多个双精度类型的数组，然后必须使用SetIndexBuffer()函数用作指标缓冲区。若要绘制指标标图，只可使用指标缓冲区的值，任何其他数组都不能为此目的使用。另外，缓冲区的值展示在数据窗口。

一个指标缓冲区应该是动态的并且不需要规范大小 - 用作指标缓冲区的数组大小，通过程序端执行子系统自动设置。

数组绑定指标缓冲区以后，标引方向 像普通数组一样默认设置，但是您可以使用ArraySetAsSeries()函数更改访问数组元素的方式。默认下，指标缓冲区被用于存储用于标图的数据 (INDICATOR_DATA)。

如果指标值的计算需要保持中间计算并存储每个柱的额外值，然后这个数组可以被声明为绑定期间的计算缓冲区 (INDICATOR_CALCULATIONS)。对于中间值，您也可以使用正规数组，但是在这种情况下，程序员必须管理数组大小。

一些标图允许为每个柱形设置颜色。若要存储颜色信息，需要使用颜色缓冲区 (INDICATOR_COLOR_INDEX)。颜色是整型color ，但是所有指标缓冲区必须是具有双精度类型的特点。颜色的值和辅助 (INDICATOR_CALCULATIONS) 缓冲区的值不能通过使用CopyBuffer()来获得。

指标缓冲区的数量必须使用编译程序指令#property indicator_buffers number_of_buffers来指定：

```
#property indicator_buffers 3  //  指标有3个缓冲区
```

一个指标可允许的最大缓冲区数量是512。

## 指标缓冲区和标图的关联

每个标图都是基于一个或多个指标缓冲区。所以，若要显示简单的蜡烛图，则需要四个值 - 开盘价，最高价，最低价和收盘价。相应地，若要以蜡烛图的形式显示一个指标，必须为它们声明4个指标缓冲区和4个双精度类型的数组。例如：

```
//--- 指标有4个指标缓冲区
#property indicator_buffers 4
//--- 指标有一个标图
#property indicator_plots   1
//--- 图形标图数1将显示为蜡烛图
#property indicator_type1   DRAW_CANDLES
//--- 蜡烛图将会以clrDodgerBlue绘制
#property indicator_color1  clrDodgerBlue
//--- 指标缓冲区的4个数组
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

© 2000-2025, MetaQuotes Ltd.

图形标图根据标图号自动使用指标缓冲区。标图数从1开始，缓冲区数从0开始。如果第一个标图需要4个指标缓冲区，那么开始的4个指标缓冲区将被用于绘制它。这四个缓冲区应该使用 SetIndexBuffer() 函数，以正确的标引连接相应的数组。

```
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,OBuffer,INDICATOR_DATA);  // 第一指标缓冲区对应0标引
   SetIndexBuffer(1,HBuffer,INDICATOR_DATA);  // 第二指标缓冲区对应1标引
   SetIndexBuffer(2,LBuffer,INDICATOR_DATA);  // 第三缓冲区对应2标引
   SetIndexBuffer(3,CBuffer,INDICATOR_DATA);  // 第四缓冲区对应3标引
```

标图蜡烛图，指标将只使用这开始的四个缓冲区，因为 "蜡烛图"的标图在第一个数字下已经公布。

改变示例，并添加简单的线型标图 - DRAW_LINE。现在假设线的编号为1，而蜡烛图的编号为2。缓冲区和标图的数量会增加。

```
//--- 指标有5个指标缓冲区
#property indicator_buffers 5
//--- 指标有2个标图
#property indicator_plots   2
//--- 图 1 是线型
#property indicator_type1   DRAW_LINE
//--- 线的颜色是 clrDodgerRed
#property indicator_color1  clrDodgerRed
//--- 图 2 是日式蜡烛图
#property indicator_type2   DRAW_CANDLES
//--- 蜡烛图的颜色是clrDodgerBlue
#property indicator_color2  clrDodgerBlue
//--- 指标缓冲区的5个数组
double LineBuffer[];
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

标图顺序已经改变，现在线型是第一位，紧跟着是日式蜡烛图。因此，缓冲区的顺序也相应改变 - 首先我们公布的是0标引线型缓冲区，然后是四个蜡烛图缓冲区。

```
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);  // 第一个缓冲区对应0标引
//--- 绑定数组和蜡烛图指标缓冲区
   SetIndexBuffer(1,OBuffer,INDICATOR_DATA);     // 第二个缓冲区对应1标引
   SetIndexBuffer(2,HBuffer,INDICATOR_DATA);     // 第三个缓冲区对应2标引
   SetIndexBuffer(3,LBuffer,INDICATOR_DATA);     // 第四个缓冲区对应3标引
   SetIndexBuffer(4,CBuffer,INDICATOR_DATA);     // 第五个缓冲区对应4标引
```

缓冲区和标图的数量仅可以通过使用编译程序指令来设置，不可以使用函数动态改变这些属性。

© 2000-2025, MetaQuotes Ltd.

## 颜色版本样式

正如表格中所见，样式被分成两组。第一组包括名字中无COLOR的样式，我们调用这些基本样式：

- DRAW_LINE
- DRAW_SECTION
- DRAW_HISTOGRAM
- DRAW_HISTOGRAM2
- DRAW_ARROW
- DRAW_ZIGZAG
- DRAW_FILLING
- DRAW_BARS
- DRAW_CANDLES

在第二组，样式名称包括 COLOR，我们称它们颜色版本：

- DRAW_COLOR_LINE
- DRAW_COLOR_SECTION
- DRAW_COLOR_HISTOGRAM
- DRAW_COLOR_HISTOGRAM2
- DRAW_COLOR_ARROW
- DRAW_COLOR_ZIGZAG
- DRAW_COLOR_BARS
- DRAW_COLOR_CANDLES

所有颜色版本的样式不同于基本版，因为他们允许为标图的每个部分指定颜色。标图的最小部分是柱形，所以我们可以说颜色版允许在每个柱形上设置颜色。

除了 DRAW_NONE 和 DRAW_FILLING样式，它们没有颜色版。

若要在每个柱上设置标图颜色，存储颜色标引的额外缓冲区已被添加到颜色版。这些标引在特定数组指出颜色数量，包括预定颜色组。颜色数组大小是64.这意味着每个样式的演示版都允许用64种不同颜色涂色标图。

在指定颜色数组中，颜色的设置和数量可以通过编译程序指令#property indicator_color来设置，在这里您可以通过逗号分隔指定所有必需的颜色。例如，指标中的这种输入：

```
//--- 定义8种颜色用于蜡烛图涂色 它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
```

说明标图　　1，8种颜色被设置，将被置于指定数组。另外在程序中，我们将不会指定标图的颜色，但不仅它的标引。如果我们想要为K柱设置红色，那么来自数组的颜色标引值应该在指标的颜色缓冲区设置。红色首先在指令中指定，它对应0标引号。

```
  //--- 设置蜡烛图颜色clrRed
  col_buffer[buffer_index]=0;
```

颜色设置不会一劳永逸的一次性给予，它可以使用PlotIndexSetInteger()函数动态变化。例如：

© 2000-2025, MetaQuotes Ltd.

```
   //--- 为每个标引设置颜色，像属性PLOT_LINE_COLOR
   PlotIndexSetInteger(0,                          //   图形样式的数量
                       PLOT_LINE_COLOR,            //   属性标识符
                       plot_color_ind,             //   颜色标引，在这里我们编写颜色
                       color_array[i]);            //   新颜色
```

## 指标和标图属性

对于指标标图，属性可以借助编译程序指令和使用相应的函数进行设置。在指标属性和函数之间的连接文章中阅读关于它的更多信息。使用指定函数动态改变指标属性允许创建更灵活的自定义指标。

## 图表上指标绘制的起点

在许多情况下，根据算法的条件，不可能在当前柱立即启动计算指标值，因为在历史记录中必须提供之前柱的最小数。例如，多种平滑类型意味着在之前的N柱上使用价格数组，并且在这些值的基础上，计算当前柱的指标值。

在这种情况下，既无法为开始的N柱计算指标值，这些值也不能显示在图表上而只能辅助计算进一步的值。为避免在历史记录的开始N柱上标图指标，则在相应的标图设置N值到PLOT_DRAW_BEGIN属性：

```
//--- 绑定数组和蜡烛图的指标缓冲区
PlotIndexSetInteger(number_of_plot,PLOT_DRAW_BEGIN,N);
```

在这里：

- number_of_plot -从0到indicator_plots-1的值（从0开始的标图数）。
- N - 历史记录中开始柱的数量，在这里指标不能显示在图表上。

© 2000-2025, MetaQuotes Ltd.

# DRAW_NONE

DRAW_NONE样式是专为需要计算缓冲区值和将其显示在数据窗口的情况下使用而设计的，但图表上的标图并不要求。若要建立精确性，请在OnInit()函数使用IndicatorSetInteger(INDICATOR_DIGITS,num_chars)表达式：

```
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- 设置显示在数据窗口的精确值
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
   return(INIT_SUCCEEDED);
  }
```

DRAW_NONE标图所需的缓冲区的数量是1。

在数据窗口，显示鼠标当前所悬停柱形数量的指标示例。编号对应时间帧，表示当前未完成的柱形有0标引，最早的柱形有最大的标引。

注意，即使是事实，因为红色设置标图 №1，指标也不会在图表上绘制任何内容。

```
//+------------------------------------------------------------------+
//|                                                    DRAW_NONE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图不可见
#property indicator_label1  "Bar Index"
#property indicator_type1   DRAW_NONE
#property indicator_style1  STYLE_SOLID
#property indicator_color1  clrRed
#property indicator_width1  1
//--- 指标缓冲区
double         InvisibleBuffer[];
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- 设置展示在数据窗口的精确值
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static datetime lastbar=0;
//--- 如果这是指标的第一次计算
   if(prev_calculated==0)
     {
      //--- 第一次重新编号柱形
      CalcValues(rates_total,close);
      //--- 在lastbar记下当前柱的开盘时间
      lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
     }
   else
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果新柱出现，其开盘时间不同于lastbar
      if(lastbar!=SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE))
        {
         //--- 再次重新编号柱形
         CalcValues(rates_total,close);
         //--- 在lastbar更新当前柱的开盘时间
         lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
        }
      }
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 编制类似时间帧的柱形编号                                         |
//+------------------------------------------------------------------+
void CalcValues(int total,double const  &array[])
  {
//--- 设置时间帧的指标缓冲区标引
   ArraySetAsSeries(InvisibleBuffer,true);
//--- 用其编号填写每个柱形
   for(int i=0;i<total;i++) InvisibleBuffer[i]=i;
  }
```

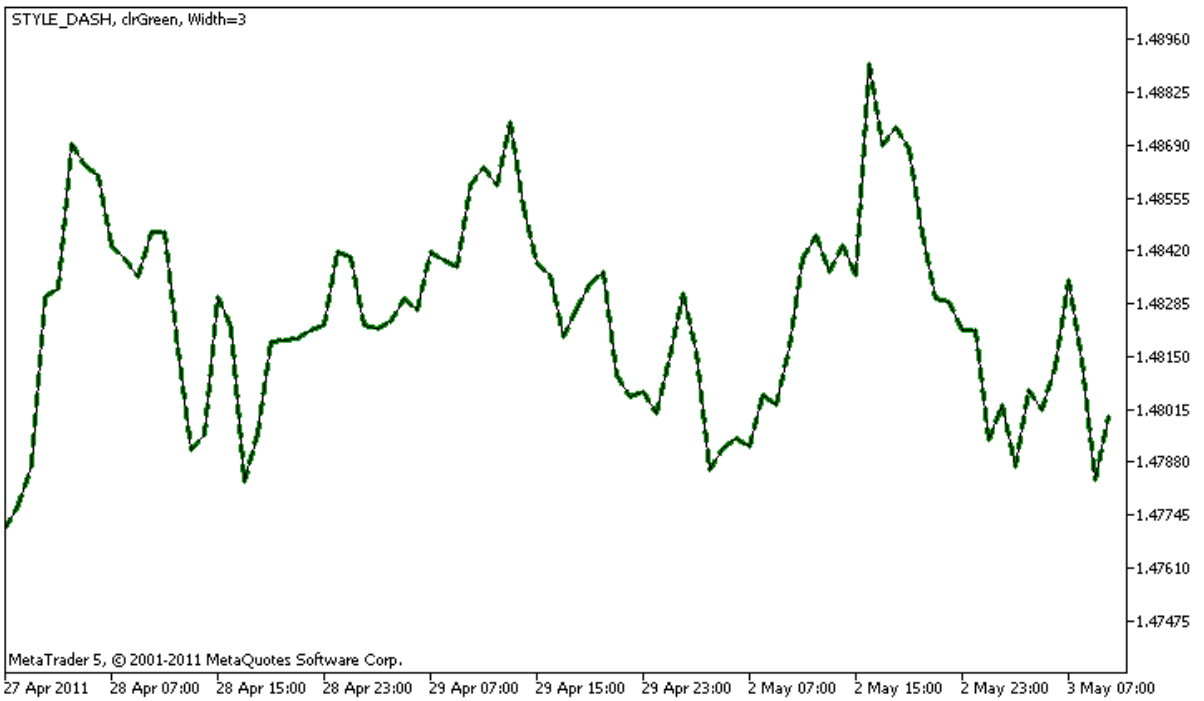
© 2000-2025, MetaQuotes Ltd.

# DRAW_LINE

DRAW_LINE通过指标缓冲区的值以指定颜色绘制一条线。线的宽度，样式和颜色可以使用编译程序指令和动态使用PlotIndexSetInteger()函数进行设定。 动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

标图DRAW_LINE所需的缓冲区数量是1。

使用柱形收盘价绘制线型的指标的示例。线型颜色，宽度和样式每N=5 个订单号都会随机变化一次。




请注意，最初DRAW_LINE 属性的**plot1** 使用编译程序指令#property设置，然后在OnCalculate()函数这三种属性随机设置。N参数设置在指标的外部参数，使手动配置成为可能 (指标属性窗口的参数标签)。

```
//+------------------------------------------------------------------+
//|                                                   DRAW_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_LINE"
#property description "It draws a line of a specified color at Close prices"
#property description "Color, width and style of lines is changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
```

```
#property indicator_buffers 1
#property indicator_plots   1
//--- 线型属性使用编译程序指令设置
#property indicator_label1  "Line"       // 数据窗口的标图名称
#property indicator_type1   DRAW_LINE    // 标图类型是线型
#property indicator_color1  clrRed       // 线型颜色
#property indicator_style1  STYLE_SOLID // 线型样式
#property indicator_width1  1            // 线型宽度
//--- 输入参数
input int       N=5;          // 改变的订单号数量
//--- 标图的指标缓冲区
double          LineBuffer[];
//--- 存储颜色的数组
color colors[]={clrRed,clrBlue,clrGreen};
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//--- 初始化随机数字生成器
   MathSrand(GetTickCount());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
```

© 2000-2025, MetaQuotes Ltd.

```
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值的模块
   for(int i=0;i<rates_total;i++)
     {
      LineBuffer[i]=close[i];
     }

//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变指标中绘制的线型外观                                         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型颜色的模块
//--- 获得随机数
   int number=MathRand();
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为PLOT_LINE_COLOR属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+(string)colors[color_index];

//--- 改变线型宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为 PLOT_LINE_WIDTH 属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+", Width="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
```

© 2000-2025, MetaQuotes Ltd.

```
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm=EnumToString(styles[style_index])+", "+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_SECTION

DRAW_SECTION 通过指标缓冲区的值以指定颜色绘制一个节段。线的宽度，颜色和样式的指定类似DRAW_LINE 样式 - 使用 编译程序指令 或动态使用PlotIndexSetInteger()函数。动态改变标图属性允许 "激活"指标，以便于可以根据当前状况改变其外观。

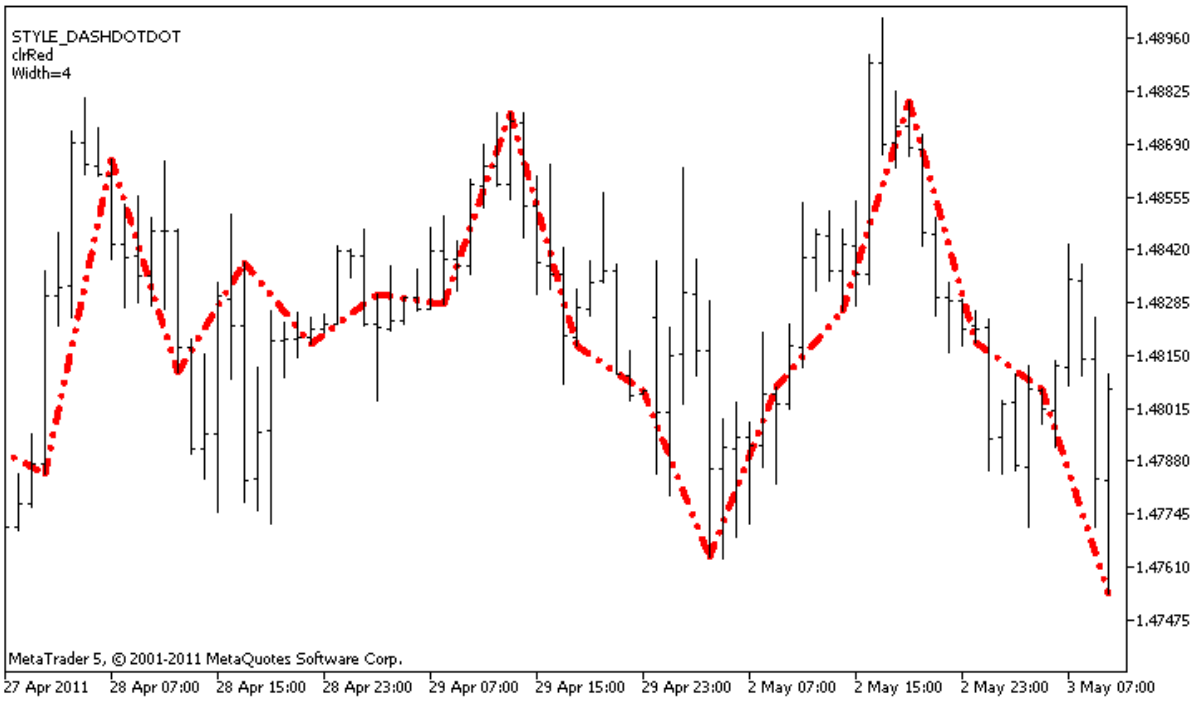
从一个非空值到指标缓冲区的另一个非空值绘制节段，空值将被忽略。若要指定哪个值应被视为 "空"，请在PLOT_EMPTY_VALUE属性设定该值：例如，如果指标应该被绘制为非零值的系列节段，然后您需要设置零值为空值：

```
//---   0 (空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，向不应标图的元素设置缓冲区空值。

标图 DRAW_SECTION所需的缓冲区的数量是 1。

绘制最高价和最低价之间节段的指标示例。所有 节段的颜色，宽度和样式每N个订单号都会随机变化一次。




请注意，最初DRAW_SECTION 属性的plot1 使用编译程序指令 #property设置，然后在OnCalculate() 这三种属性随机设置。N参数设置在指标的 外部参数 ，使手动配置成为可能 (指标属性窗口的参数标签)。

```
//+------------------------------------------------------------------+
//|                                                DRAW_SECTION.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "An indicator to demonstrate DRAW_SECTION"
#property description "Draws straight sections every bars bars"
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图节段
#property indicator_label1  "Section"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      bars=5;           // 柱形节段长度
input int      N=5;              // 改变节段样式的订单号数量
//--- 标图的指标缓冲区
double         SectionBuffer[];
//--- 计算阶段终点的辅助变量
int            divider;
//--- 存储颜色的数组
color colors[]={clrRed,clrBlue,clrGreen};
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,SectionBuffer,INDICATOR_DATA);
//--- 0（空）值将不参与绘制
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- 检查指标参数
   if(bars<=0)
     {
      PrintFormat("Invalid value of parameter bar=%d",bars);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=2*bars;
//---+
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
```

© 2000-2025, MetaQuotes Ltd.

```
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 指标值计算开始的柱数
   int start=0;
//--- 如果之前计算了指标，那么在前一柱开始设置
   if(prev_calculated>0) start=prev_calculated-1;
//--- 这是指标值的全部计算
   for(int i=start;i<rates_total;i++)
     {
      //--- 获得柱数除以2*bars的余数
      int rest=i%divider;
      //--- 如果柱数可被2*bars除尽
      if(rest==0)
        {
         //--- 在该柱最高价设置节段终点
         SectionBuffer[i]=high[i];
        }
      //---如果除法的余数等于柱数，
      else
        {
         //--- 在该柱最高价设置节段终点
         if(rest==bars) SectionBuffer[i]=low[i];
         //--- 如果什么也没发生，忽略柱形 - 设置0
         else SectionBuffer[i]=0;
        }
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 改变指标的节段外观                                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为PLOT_LINE_COLOR属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];

//--- 改变线型宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;    // 宽度设置从 0 到 4
//--- 设置宽度
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置线型样式
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

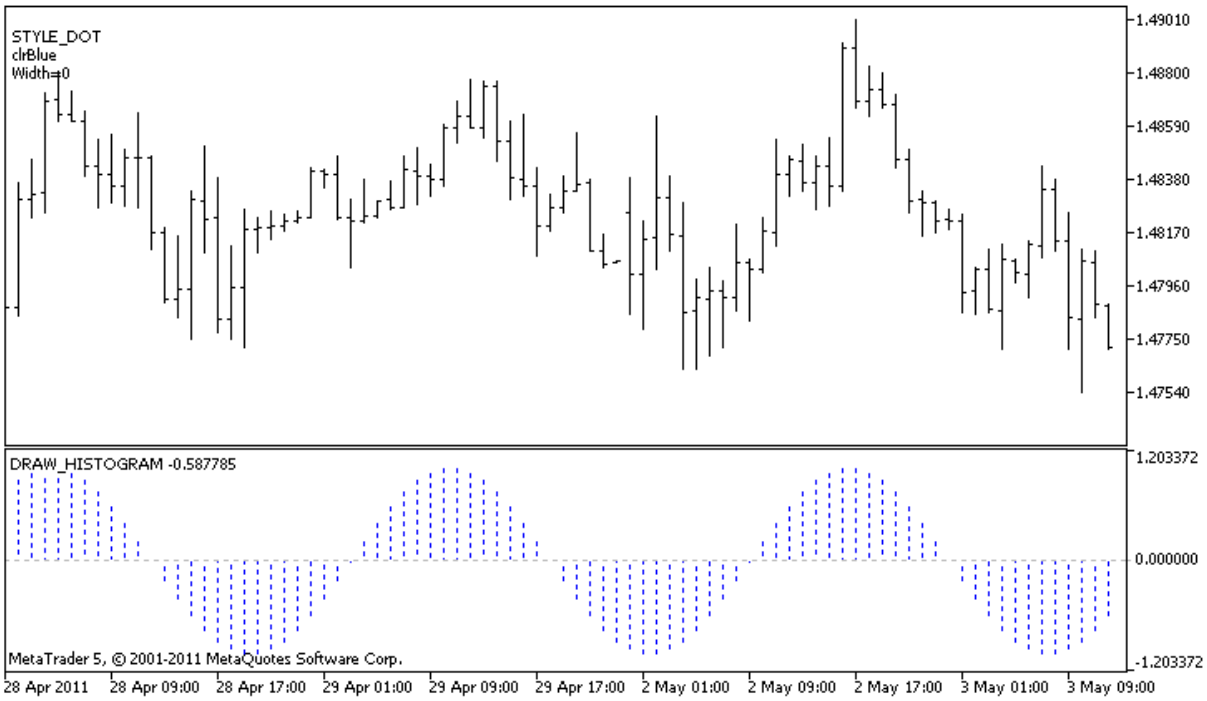
# DRAW_HISTOGRAM

DRAW_HISTOGRAM 样式根据一系列从0到指定值的指定颜色的列绘制一个直方图。该值从指标缓冲区获得。列的宽度，颜色和样式的指定类似 DRAW_LINE 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变直方图的外观。

因为每个柱上从0水平绘制列，DRAW_HISTOGRAM应该更好的用在单独的图表窗口。大部分情况下这种类型的标图用于创建oscillator类型的指标，例如，Bears Power 或 OsMA。对于空的不显示的值，应该指定0值。

标图 DRAW_HISTOGRAM 所需的缓冲区的数量是 1。

基于MathSin() 函数绘制指定颜色的正弦曲线的指标示例。所有直方图列的颜色，宽度和样式每N个订单号都会随机变化一次。柱形参数指定正弦曲线周期，指定柱形数量之后，正弦曲线将重复这个循环。




Note that initially for plot1 请注意，最初DRAW_HISTOGRAM 属性的plot1 使用编译程序指令 #property设置，然后在 OnCalculate() 函数这三种属性随机设置。N参数设置在指标的 外部参数，使手动配置成为可能 (指标属性窗口的参数标签)。

```
//+------------------------------------------------------------------+
//|                                                DRAW_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars in the cycle of the

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图直方图
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      bars=30;          // 柱形的正弦曲线周期
input int      N=5;              // 改变直方图的订单号数量
//--- 指标缓冲区
double         HistogramBuffer[];
//--- 获得2Pi角弧度的因素，当乘以柱的参数时
double    multiplier;
//--- 存储颜色的数组
color colors[]={clrRed,clrBlue,clrGreen};
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- 计算乘数
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Set the value of bars=%d greater than 1",bars);
      //--- 提前终止指标
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值
   int start=0;
//--- 如果之前OnCalculate开始期间已经计算
   if(prev_calculated>0) start=prev_calculated-1; // 设置倒数第二柱为计算起点
//--- 填写指标缓冲区的值
   for(int i=start;i<rates_total;i++)
     {
      HistogramBuffer[i]=sin(i*multiplier);
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变指标的线型外观                                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为PLOT_LINE_COLOR属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 改变线型宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;    // 宽度设置从 0 到 4
//--- 设置宽度
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置线型样式
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_HISTOGRAM2

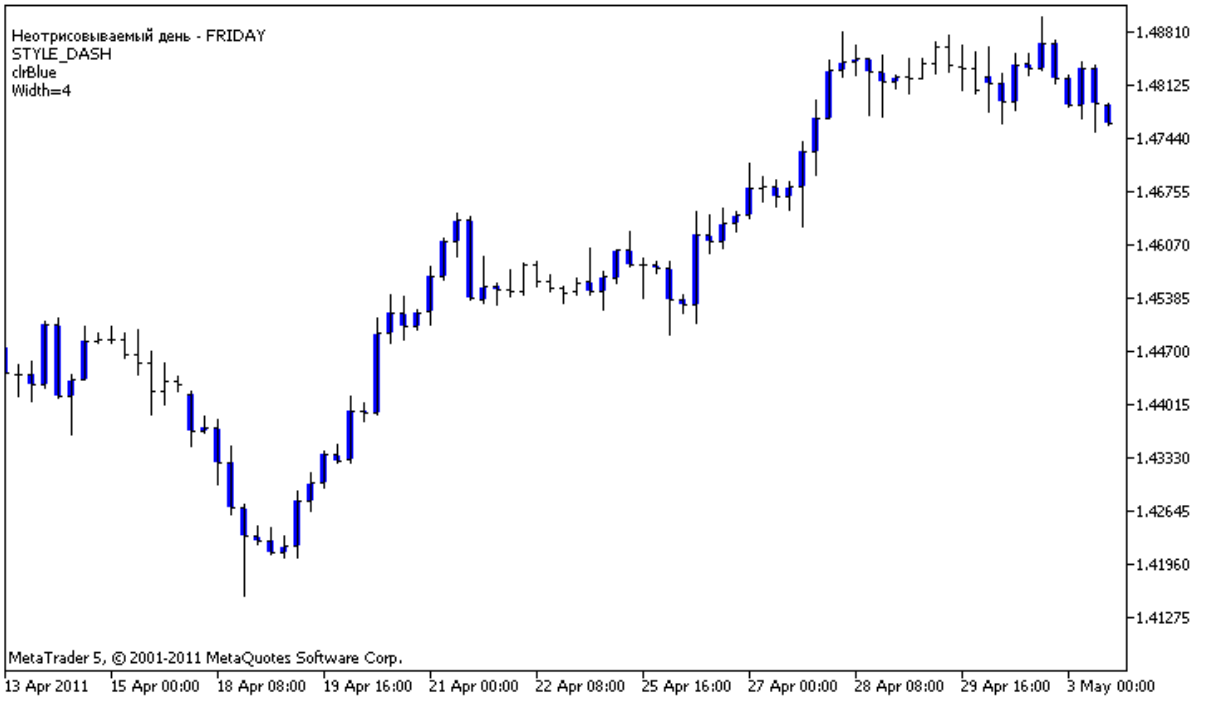
DRAW_HISTOGRAM2 样式使用两个指标缓冲区的值绘制一个指定颜色垂直段的直方图。节段的宽度，颜色和样式的指定类似DRAW_LINE 样式 - 使用编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变直方图的外观。

DRAW_HISTOGRAM2 样式可以被用在单独的图表子窗口和它的主窗口。由于不绘制空值，指标缓冲区的所有值都需要明确设定。缓冲区不通过零值初始化。

标图DRAW_HISTOGRAM2所需的缓冲区的数量是2。

在每个柱形的开盘价和收盘价之间标图指定颜色和宽度的垂直节段的指标示例。所有 直方图列的颜色，宽度和样式每N个订单号都会随机变化一次。指标启动期间，在OnInit() 函数，不会绘制直方图的工作日数量 - invisible_day - 随机设置。为此，空值设置 PLOT_EMPTY_VALUE=0:

```
//--- 设置空值
   PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```




请注意，最初DRAW_HISTOGRAM2属性的plot1 使用编译程序指令 #property设置，然后在OnCalculate()函数这三种属性随机设置。N 参数设置在指标的 外部参数 ，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                                DRAW_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "An indicator to demonstrate DRAW_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图Histogram_2
#property indicator_label1  "Histogram_2"
#property indicator_type1   DRAW_HISTOGRAM2
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      N=5;              // 改变直方图的订单号数量
//--- 指标缓冲区
double         Histogram_2Buffer1[];
double         Histogram_2Buffer2[];
//--- 没有标图指标的工作日
int invisible_day;
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen};
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,Histogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Histogram_2Buffer2,INDICATOR_DATA);
//--- 设置空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- 获得从0到5的随机数
   invisible_day=MathRand()%6;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值
   int start=0;
//--- 通过每柱的开盘价获得工作日
   MqlDateTime dt;
//---如果之前OnCalculate开始期间已经计算
   if(prev_calculated>0) start=prev_calculated-1; // 在倒数第二柱设置计算起点
//--- 填写指标缓冲区的值
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
      if(dt.day_of_week==invisible_day)
        {
         Histogram_2Buffer1[i]=0;
         Histogram_2Buffer2[i]=0;
        }
      else
        {
         Histogram_2Buffer1[i]=open[i];
         Histogram_2Buffer2[i]=close[i];
        }
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变指标的线型外观                                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 线型颜色变化模块
```

© 2000-2025, MetaQuotes Ltd.

```
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];

//--- 改变线型宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 宽度设置从 0 到 4
//--- 设置线型宽度
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置线型样式
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- 添加计算中省略的工作日的信息
   comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_ARROW

DRAW_ARROW 样式根据指标缓冲区的值绘制指定颜色的箭头（Wingdings的设置符号）。交易品种的宽度和颜色的指定类似 DRAW_LINE 样式 - 使用编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变指标的外观。

交易品种代码使用PLOT_ARROW 属性进行设置。

```
//--- 定义PLOT_ARROW绘制的来自Wingdings字体的符号代码到
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

PLOT_ARROW 的默认值=159（一圈）。

每个箭头实际上都是有高度和定位点的符号，可以包含图表上的一些重要信息（例如，柱形的收盘价）。因此，我们可以额外指定像素的垂直移动，这并不依赖图表的比例。箭头沿着指定的像素数量将向下移动，虽然指标的值将会保持相同：

```
//--- 设置箭头的像素垂直移动
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

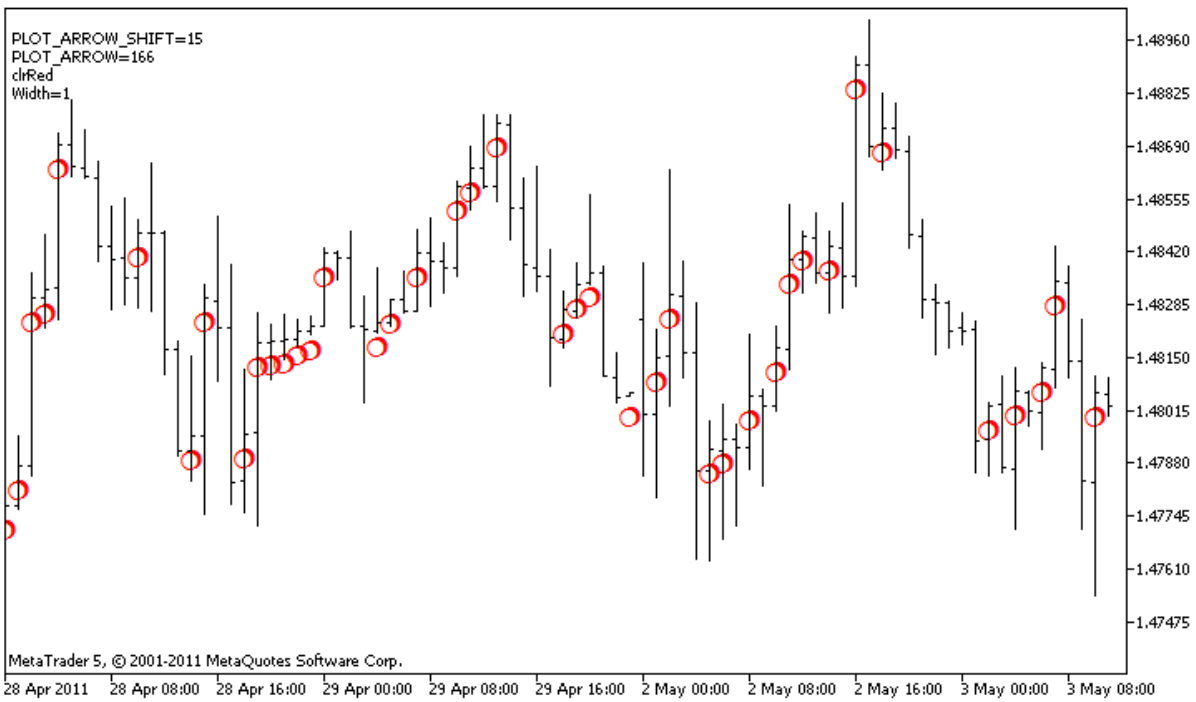
PLOT_ARROW_SHIFT的负值意味着箭头向上移动，正值表示箭头向下移动。

DRAW_ARROW 样式可以被用在单独的图表子窗口和它的主窗口。由于不绘制空值并且也不会出现在"数据窗口"，指标缓冲区的所有值都应该明确设定。缓冲区不通过零值初始化。

```
//--- 设置空值
   PlotIndexSetDouble(index_of_plot_DRAW_ARROW,PLOT_EMPTY_VALUE,0);
```

标图DRAW_ARROW 所需的缓冲区的数量是 1。

以比前柱收盘价高的收盘价在每柱上绘制箭头的指标示例。所有 箭头的颜色，宽度，移动和符号代码每 N 个订单号都会随机变化一次。




© 2000-2025, MetaQuotes Ltd.

在示例中，对于DRAW_ARROW样式的plot1，属性，颜色和大小使用编译程序指令#property设置，然后在OnCalculate()函数随机设置属性。N参数设置在指标的外部参数，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                                   DRAW_ARROW.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ARROW"
#property description "Draws arrows set by Unicode characters, on a chart"
#property description "The color, size, shift and symbol code of the arrow are changed
#property description "after every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图箭头
#property indicator_label1  "Arrows"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrGreen
#property indicator_width1  1
//--- 输入参数
input int      N=5;          // 改变订单号数量
input ushort   code=159;     // 在DRAW_ARROW绘制的符号代码
//--- 标图的指标缓冲区
double         ArrowsBuffer[];
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen};
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,ArrowsBuffer,INDICATOR_DATA);
//--- 定义PLOT_ARROW绘制的符号代码
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- 设置箭头的像素垂直移动
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- 设置空值为 0
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变颜色，大小，移动和箭头的代码
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值的模块
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- 计算循环
   for(int i=1;i<rates_total;i++)
     {
      //--- 如果当前收盘价高于之前的价格，请绘制一个箭头
      if(close[i]>close[i-1])
         ArrowsBuffer[i]=close[i];
      //--- 否则指定零值
      else
         ArrowsBuffer[i]=0;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变指标的交易品种外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成指标属性信息的字符串
```

© 2000-2025, MetaQuotes Ltd.

```
   string comm="";
//--- 改变箭头颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];

//--- 改变大小箭头的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 大小设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下箭头大小
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变箭头代码 (PLOT_ARROW)的模块
   number=MathRand();
//--- 获得整数除法的余数来计算新的箭头代码 从0到19)
   int code_add=number%20;
//--- 设置新的符号代码作为ode+code_add的结果
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- 写下符号代码 PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- 改变箭头的像素垂直移动的模块
   number=MathRand();
//--- 获得整数除法余数的移动
   int shift=20-number%41;
//--- 设置新移动从 -20 到 20
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- 写下PLOT_ARROW_SHIFT移动
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_ZIGZAG

DRAW_ZIGZAG 样式根据指标缓冲区的值绘制指定颜色的节段。该样式非常相似于 DRAW_SECTION，但与后者都不太相同，如果为该柱设置两个指标缓冲区的值，它允许在一个柱内绘制垂直节段。节段从第一个缓冲区的值标图到第二个指标缓冲区的值。缓冲区不会只包含空值，因为在这种情况下，不会标图。

线的宽度，颜色和样式的指定类似 DRAW_SECTION 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许 "激活" 指标，以便于可以根据当前状况改变其外观。

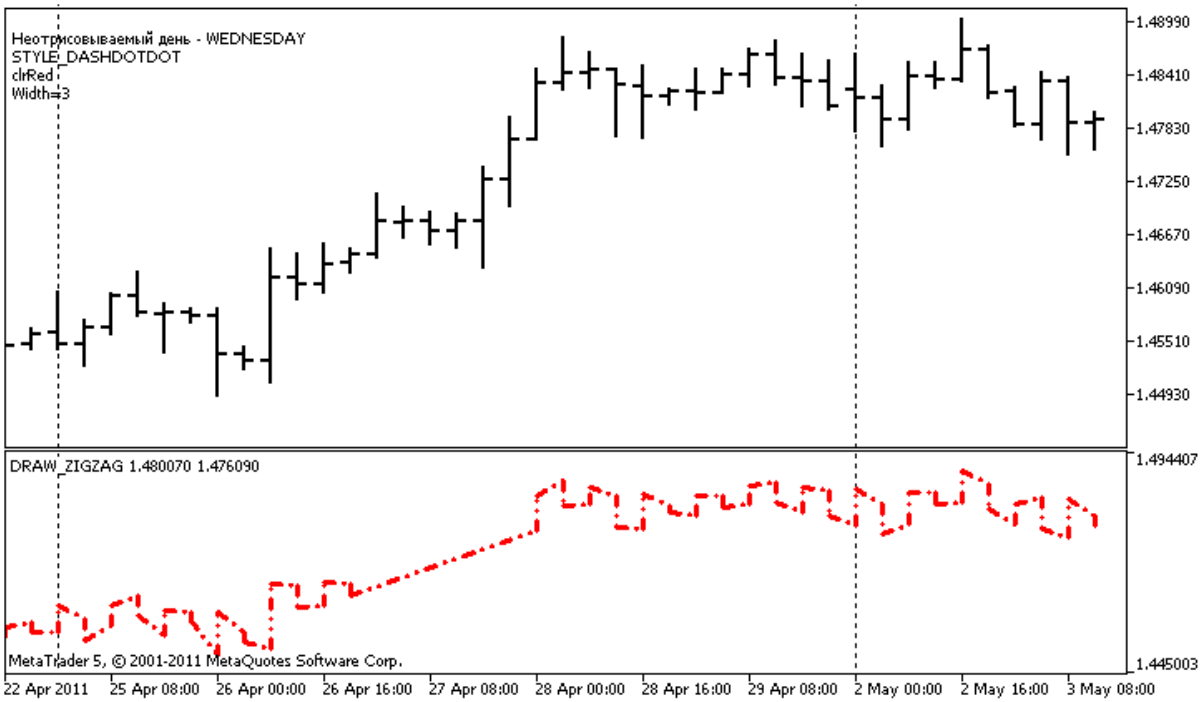
从一个缓冲区的非空值到另一个指标缓冲区的非空值绘制部分。若要指定哪个值应被视为 "空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_ZIGZAG所需的缓冲区的数量是2。

基于最高价和最低价标图的指标示例。zigzag 线的颜色，宽度和样式每N 个订单号都会随机变化一次。




请注意，最初DRAW_ZIGZAG属性的plot1 使用编译程序指令#property设置，然后在OnCalculate()函数这些属性随机设置。N 参数设置在指标的 外部参数，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                                  DRAW_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ZIGZAG"
#property description "It draws a \"saw\" as straight segments, skipping the bars of c
#property description "The day to skip is selected randomly during indicator start"
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图ZigZag
#property indicator_label1  "ZigZag"
#property indicator_type1   DRAW_ZIGZAG
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      N=5;              // 改变订单号数量
//--- 指标缓冲区
double         ZigZagBuffer1[];
double         ZigZagBuffer2[];
//--- 没有标图指标的工作日
int invisible_day;
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen};
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                                |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,ZigZagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ZigZagBuffer2,INDICATOR_DATA);
//--- 获得从0到6的随机值，在这一天不标图指标
   invisible_day=MathRand()%6;
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- 0(空)值将不参与绘制
   PlotIndexSetString(0,PLOT_LABEL,"ZigZag1;ZigZag2");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                  |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 获得每个柱形工作日所需的时间结构
   MqlDateTime dt;

//--- 计算的初始位置
   int start=0;
//--- 如果在前一个订单号计算指标，那么在倒数第二个开始计算
   if(prev_calculated!=0) start=prev_calculated-1;
//--- 计算循环
   for(int i=start;i<rates_total;i++)
     {
      //--- 在结构中写下柱形的开盘时间
      TimeToStruct(time[i],dt);
      //--- 如果该柱的工作日等于invisible_day
      if(dt.day_of_week==invisible_day)
        {
         //--- 写下该柱形的缓冲区空值
         ZigZagBuffer1[i]=0;
         ZigZagBuffer2[i]=0;
        }
      //--- 如果工作日没问题，填写缓冲区
      else
        {
         //--- 如果柱形数量
         if(i%2==0)
```

© 2000-2025, MetaQuotes Ltd.

```
            {
             //---   在第一个缓冲区写下最高价，在第二个缓冲区写下最低价
             ZigZagBuffer1[i]=high[i];
             ZigZagBuffer2[i]=low[i];
            }
          //--- 柱形数量是奇数
          else
            {
             //--- 按相反顺序填写柱形
             ZigZagBuffer1[i]=low[i];
             ZigZagBuffer2[i]=high[i];
            }
         }
      }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变zigzag节段的外观                                             |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成ZigZag属性信息的字符串
   string comm="";
//--- 改变ZigZag颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];

//--- 改变线型宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
```

© 2000-2025, MetaQuotes Ltd.

```
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- 添加计算中省略的工作日的信息
   comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_FILLING

DRAW_FILLING 样式在两个指标缓冲区之间标图颜色区域。实际上，该样式绘制两条线，以指定颜色二选一，填充它们之间的空间。它用于创建绘制通道的指标。缓冲区不会只包含空值，因为在这种情况下，不会标图。

您可以设置两种填充颜色：

- 第一个颜色用于第一个缓冲区的值比第二个指标缓冲区的值大的区域；
- 第二个颜色用于第二个缓冲区的值大于第一个指标缓冲区的值大的区域。

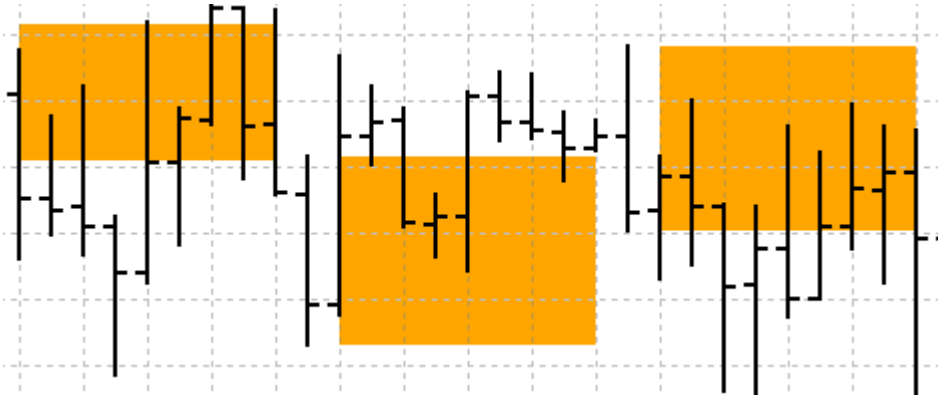
填充颜色的设置可以使用 编译程序指令 或动态使用PlotIndexSetInteger() 函数。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

该指标用于计算所有的柱，因此两个指标缓冲区的值既不等于0也不等于空值。若要指定哪个值应被视为"空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

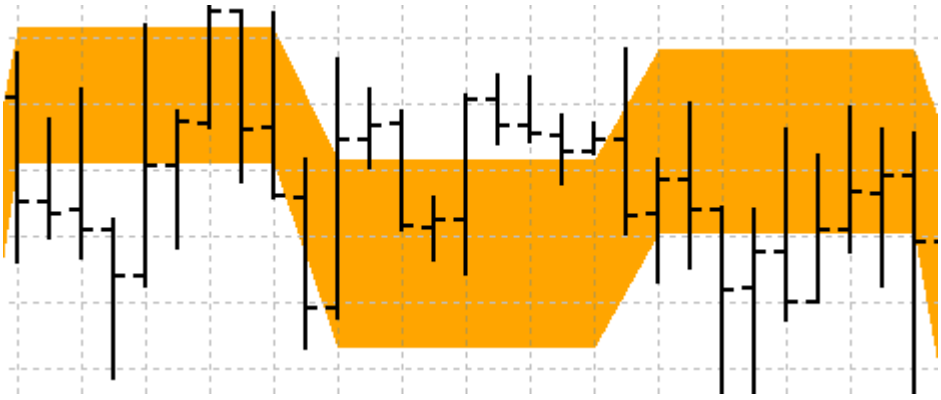
```
   #define INDICATOR_EMPTY_VALUE -1.0
   ...
//--- INDICATOR_EMPTY_VALUE (空值) 将不参与计算
   PlotIndexSetDouble (DRAW_FILLING_creation_index,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VA
```

不参与指标计算的柱形绘制将取决于指标缓冲区的值：

- 两个指标缓冲区的值都等于0的柱形，不参与绘制指标。这意味着零值的区域不填充。



- 指标缓冲区的值等于"空值"的柱形，参与绘制指标。空值区域将被填充，以便连接有重要值的区域。

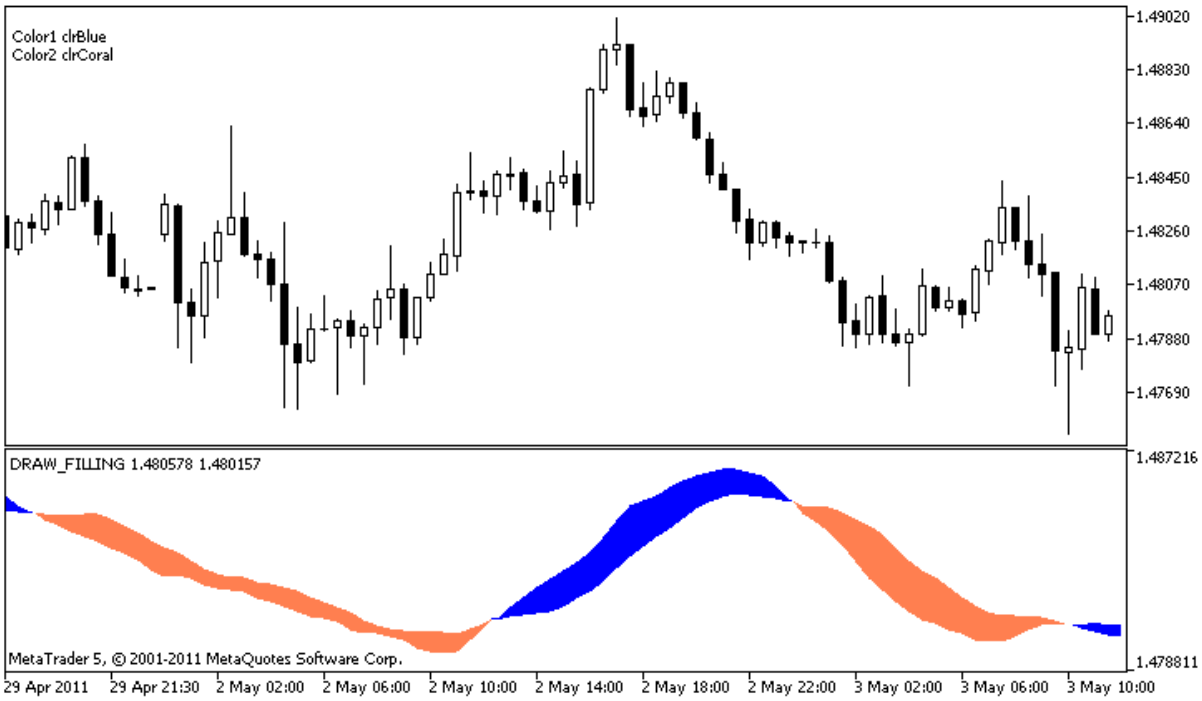


应该注意，如果"空值"等于零，不参与指标计算的柱形也会被填充。

© 2000-2025, MetaQuotes Ltd.

标图DRAW _FILLING 所需的缓冲区的数量是 2。

在独立窗口以不同的平均周期在两个MAs之间绘制通道的指标示例。在移动平均数视觉交叉的颜色变化显示了向上向下趋势的变化。每N个订单号都会随机变化一次颜色。N 参数设置在指标的 外部参数，使手动配置成为可能（指标属性窗口的参数标签）。




请注意，最初DRAW _FILLING属性的plot1 使用编译程序指令#property设置，然后在 OnCalculate() 函数新颜色随机设置。

```
//+------------------------------------------------------------------+
//|                                                DRAW_FILLING.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_FILLING"
#property description "It draws a channel between two MAs in a separate window"
#property description "The fill color is changed randomly"
#property description "after every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图交集
#property indicator_label1  "Intersection"
#property indicator_type1   DRAW_FILLING
#property indicator_color1  clrRed,clrBlue
```

```
#property indicator_width1  1
//--- 输入参数
input int       Fast=13;          // 快速MA的周期
input int       Slow=21;          // 慢速MA的周期
input int       shift=1;          // 向未来的MAs转移 正值）
input int       N=5;              // 改变订单号数量
//--- 指标缓冲区
double          IntersectionBuffer1[];
double          IntersectionBuffer2[];
int fast_handle;
int slow_handle;
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen,clrAquamarine,clrBlanchedAlmond,clrBrown,clrC
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,IntersectionBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,IntersectionBuffer2,INDICATOR_DATA);
//---
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//---
   fast_handle=iMA(_Symbol,_Period,Fast,0,MODE_SMA,PRICE_CLOSE);
   slow_handle=iMA(_Symbol,_Period,Slow,0,MODE_SMA,PRICE_CLOSE);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 重置0计数器
      ticks=0;
     }

//--- 指标的第一次计算，或数据变化，需要一次完全重新计算
   if(prev_calculated==0)
     {
      //--- 复制所有指标值至相应的缓冲区
      int copied1=CopyBuffer(fast_handle,0,0,rates_total,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,rates_total,IntersectionBuffer2);
     }
   else // 仅填充那些更新的数据
     {
      //--- 获得OnCalculate()当前和之前启动之间的柱形不同
      int to_copy=rates_total-prev_calculated;
      //--- 如果没有不同，我们仍然会复制一个值 - 在零柱
      if(to_copy==0) to_copy=1;
      //--- 复制_copy 值到指标缓冲区的最末端
      int copied1=CopyBuffer(fast_handle,0,0,to_copy,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,to_copy,IntersectionBuffer2);
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变通道填充颜色                                                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);

//--- 获得选择新颜色作为整数除法余数的标引
   int color_index1=number%size;
//--- 设置第一个颜色为PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,colors[color_index1]);
//--- 写下第一个颜色
   comm=comm+"\r\nColor1 "+(string)colors[color_index1];

//--- 获得选择新颜色作为整数除法余数的标引
   number=MathRand(); // 获得随机数
   int color_index2=number%size;
//--- 设置第二个颜色为PLOT_LINE_COLOR 属性
```

© 2000-2025, MetaQuotes Ltd.

```
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,colors[color_index2]);
//---写下第二个颜色
   comm=comm+"\r\nColor2 "+(string)colors[color_index2];
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_BARS

DRAW_BARS 样式绘制四个指标缓冲区值的柱形，包括开盘价，最高价，最低价和收盘价。它用于创建自定义指标为柱形，包括单独的图表子窗口中和其他金融工具上的指标。

柱形颜色的设置可以使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

指标仅向那些柱形绘制，为此，设置所有四个指标缓冲区的非空值。 若要指定哪个值应被视为"空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_BARS,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_BARS所需的缓冲区的数量是4。标图的所有缓冲区都应该按照给定的顺序一个接一个地进行：开盘价，最高价，最低价和收盘价。缓冲区不会只包含空值，因为在这种情况下，不会标图。

在单独窗口的选定金融工具上绘制柱形的指标示例。柱形颜色每 N个订单号都会随机变化一次。N 参数设置在指标的 外部参数，使手动配置成为可能（指标属性窗口的参数标签）。

请注意，DRAW_BARS样式的**plot1**，使用编译程序指令 #property设置颜色，然后在 OnCalculate() 函数，从早期已备列表随机设置颜色。

```
//+------------------------------------------------------------------+
//|                                                   DRAW_BARS.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
```

© 2000-2025, MetaQuotes Ltd.

```
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_BARS"
#property description "It draws bars of a selected symbol in a separate window"
#property description "The color and width of bars, as well as the symbol are changed
#property description "every N ticks"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- 标图柱形
#property indicator_label1  "Bars"
#property indicator_type1   DRAW_BARS
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      N=5;              // 改变类型的订单号数量
input int      bars=500;         // 显示的柱形数量
input bool     messages=false;   // 在"EA交易"日志显示信息
//--- 指标缓冲区
double         BarsBuffer1[];
double         BarsBuffer2[];
double         BarsBuffer3[];
double         BarsBuffer4[];
//--- 交易品种名称
string symbol;
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 如果柱形非常小 - 提前完成工作
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- 指标缓冲区映射
   SetIndexBuffer(0,BarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,BarsBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,BarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,BarsBuffer4,INDICATOR_DATA);
//--- 绘制柱形的交易品种名称
   symbol=_Symbol;
//--- 设置交易品种的展示
```

© 2000-2025, MetaQuotes Ltd.

```
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+symbol+")");
//--- 空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式,颜色和线型宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 选择来自市场报价窗口的新交易品种
      symbol=GetRandomSymbolName();
      //--- 改变线型属性
      ChangeLineAppearance();

      int tries=0;
      //--- 试图5 次填充缓冲区的交易品种价格
      while(!CopyFromSymbolToBuffers(symbol,rates_total) && tries<5)
        {
         //--- CopyFromSymbolToBuffers() 函数调用的计数器
         tries++;
        }
      //--- 重置0计数器
      ticks=0;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充指标缓冲区的价格                                             |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 在rates[] 数组中，我们将复制开盘价，最高价，最低价和收盘价
   MqlRates rates[];
//--- 尝试计数器
   int attempts=0;
//--- 已复制多少
   int copied=0;
//--- 试图25次获得所需交易品种的时间帧
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- 如果复制足够数量的柱形失败
   if(copied!=bars)
     {
      //--- 形成信息字符串
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- 在主图表窗口的注释中显示信息
      Comment(comm);
      //--- 显示信息
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- 设置交易品种的展示
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+name+")");
     }
//--- 初始化空值缓冲区
   ArrayInitialize(BarsBuffer1,0.0);
   ArrayInitialize(BarsBuffer2,0.0);
   ArrayInitialize(BarsBuffer3,0.0);
   ArrayInitialize(BarsBuffer4,0.0);
//--- 复制缓冲区的价格
   for(int i=0;i<copied;i++)
     {
      //--- 计算缓冲区相应的标引
      int buffer_index=total-copied+i;
      //--- 写下缓冲区的价格
      BarsBuffer1[buffer_index]=rates[i].open;
      BarsBuffer2[buffer_index]=rates[i].high;
      BarsBuffer3[buffer_index]=rates[i].low;
```

© 2000-2025, MetaQuotes Ltd.

```
      BarsBuffer4[buffer_index]=rates[i].close;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| 随机返回来自市场报价的交易品种                                   |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- 市场报价窗口中显示的交易品种数量
   int symbols=SymbolsTotal(true);
//--- 列表中的交易品种位置 - 从0到交易品种的随机号
   int number=MathRand()%symbols;
//--- 返回指定位置的交易品种名称
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| 改变柱形的外观                                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成柱形属性信息的字符串
   string comm="";
//---改变柱形颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下线型颜色
   comm=comm+"\r\n"+(string)colors[color_index];

//--- 改变柱形宽度的模块
   number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 写下交易品种名称
   comm="\r\n"+symbol+comm;

//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# DRAW_CANDLES

DRAW_CANDLES 样式绘制四个指标缓冲区值的蜡烛图，包括开盘价，最高价，最低价和收盘价。它用于创建自定义指标为蜡烛图系列，包括单独的图表子窗口中和其他金融工具上的指标。

蜡烛图颜色的设置可以使用编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

指标仅向那些柱形绘制，为此，设置所有四个指标缓冲区的非空值。 若要指定哪个值应被视为"空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_CANDLES,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_CANDLES 所需的缓冲区的数量是4。标图的所有缓冲区都应该按照给定的顺序一个接一个地进行：开盘价，最高价，最低价和收盘价。缓冲区不会只包含空值，因为在这种情况下，不会标图。

您可以为DRAW_CANDLES 风格设置三种颜色来影响蜡烛图的视觉效果。如果只设置一种颜色，那么它将会应用于图表上的所有蜡烛图。

```
//--- 将单一颜色的相同蜡烛图应用于它们
#property indicator_label1  "One color candles"
#property indicator_type1   DRAW_CANDLES
//--- 仅指定一种颜色，因此所有蜡烛图都是相同颜色
#property indicator_color1  clrGreen
```

如果用逗号隔开指定两种颜色，第一种会被应用于蜡烛图外框，而第二种则应用于主体。

```
//--- 不同颜色的蜡烛图和烛芯
#property indicator_label1  "Two color candles"
#property indicator_type1   DRAW_CANDLES
//--- 绿色应用于烛芯和外框，而白色应用于主体
#property indicator_color1  clrGreen,clrWhite
```

用逗号隔开指定三种颜色，从而使上升和下降的蜡烛图可以不同的显示。在这种情况下，第一种颜色应用于蜡烛图的外框，而第二种和第三种颜色 -应用于牛市蜡烛图和熊市蜡烛图。

```
//--- 不同颜色的蜡烛图和烛芯
#property indicator_label1  "One color candles"
#property indicator_type1   DRAW_CANDLES
//--- 烛芯和外框是绿色，牛市蜡烛图的主体是白色，而熊市蜡烛图的主体是红色
#property indicator_color1  clrGreen,clrWhite,clrRed
```

因此，DRAW_CANDLES 风格允许您创建定制颜色选项的蜡烛图。除此之外，在使用PlotIndexSetInteger函数进行指标操作期间还可以更改所有颜色(composition_index_DRAW_CANDLES, PLOT_LINE_COLOR, modifier_index, colr)，在这里modifier_index 可能是以下的值：

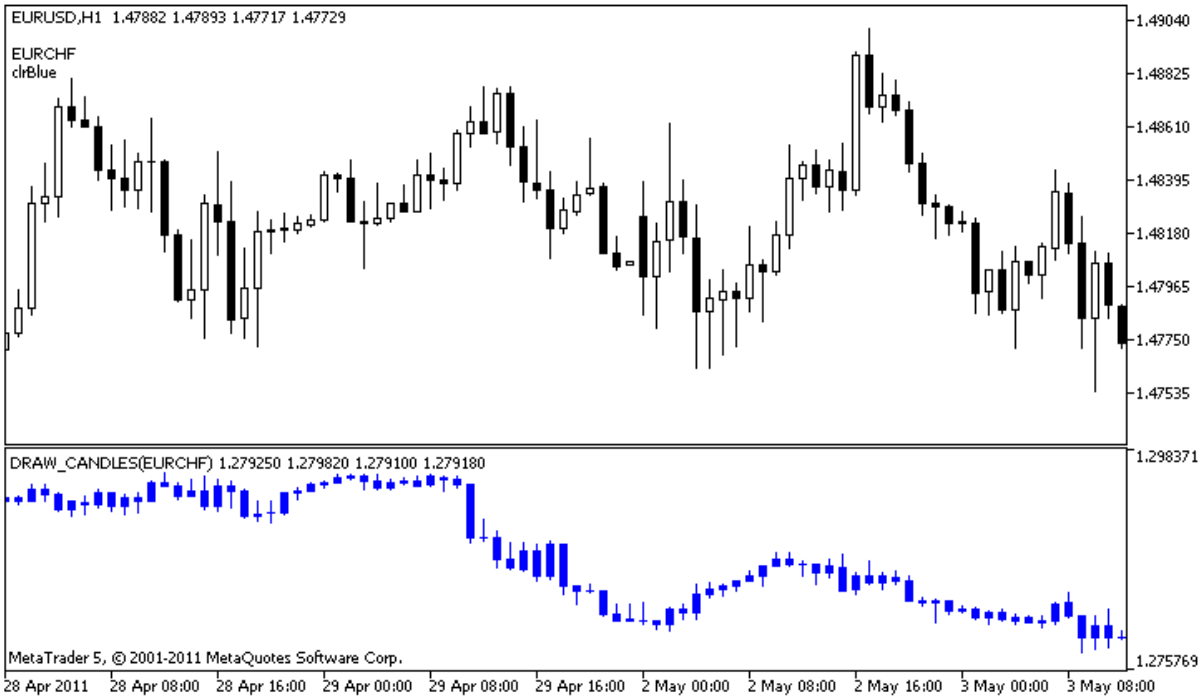
- 0 -外框和烛芯的颜色
- 1 -牛市蜡烛图主体颜色

© 2000-2025, MetaQuotes Ltd.

- 2 -熊市蜡烛图主体颜色

```
//--- 设置外框和烛芯的颜色
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrBlue);
//--- 设置牛市蜡烛图主体颜色
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrGreen);
//--- 设置熊市蜡烛图主体颜色
PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrRed);
```

在单独窗口的选定金融工具上绘制蜡烛图的指标示例。蜡烛图颜色每**N** 个订单号都会随机变化一次。N 参数设置在指标的 外部参数，使手动配置成为可能（指标属性窗口的参数标签）。




请注意，**plot1**，使用编译程序指令#property设置颜色，然后在OnCalculate() 函数，从早期已备列表随机设置颜色。

```
//+------------------------------------------------------------------+
//|                                                  DRAW_CANDLES.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate windo
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are
#property description "randomly every N ticks"

#property indicator_separate_window
```

```
#property indicator_buffers 4
#property indicator_plots   1
//--- 标图柱形
#property indicator_label1  "DRAW_CANDLES1"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- 输入参数
input int      N=5;              // 改变类型的订单号数量
input int      bars=500;         // 显示的柱形数量
input bool     messages=false;   // 在"EA交易"日志显示信息
//--- 指标缓冲区
double         Candle1Buffer1[];
double         Candle1Buffer2[];
double         Candle1Buffer3[];
double         Candle1Buffer4[];
//--- 交易品种名称
string symbol;
//--- 存储颜色的数组0到5的
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 如果柱形非常小 - 提前完成工作
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- 指标缓冲区映射
   SetIndexBuffer(0,Candle1Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Candle1Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Candle1Buffer3,INDICATOR_DATA);
   SetIndexBuffer(3,Candle1Buffer4,INDICATOR_DATA);
//--- 空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- 绘制柱形的交易品种名称
   symbol=_Symbol;
//--- 设置交易品种的展示
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_CANDLES("+symbol+")");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=INT_MAX-100;
//--- 计算订单号改变样式,颜色和线型宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 选择来自市场报价窗口的新交易品种
      symbol=GetRandomSymbolName();
      //--- 改变格式
      ChangeLineAppearance();
      //--- 选择来自市场报价窗口的新交易品种
      int tries=0;
      //--- 试图5 次填充缓冲区plot1的交易品种价格
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            Candle1Buffer1,Candle1Buffer2,Candle1Buffer3,Candle1Buffer4)
            && tries<5)
        {
         //--- CopyFromSymbolToBuffers() 函数调用的计数器
         tries++;
        }
      //--- 重置0计数器
      ticks=0;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充指定蜡烛图                                                   |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[]
```

© 2000-2025, MetaQuotes Ltd.

```
                                  )
  {
//--- 在rates[] 数组中，我们将复制开盘价，最高价，最低价和收盘价
   MqlRates rates[];
//--- 尝试计数器
   int attempts=0;
//--- 已复制多少
   int copied=0;
//--- 试图25次获得所需交易品种的时间帧
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- 如果复制足够数量的柱形失败
   if(copied!=bars)
     {
      //--- 形成信息字符串
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- 在主图表窗口的注释中显示信息
      Comment(comm);
      //--- 显示信息
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- 设置交易品种的展示
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
     }
//--- 初始化空值缓冲区
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
   ArrayInitialize(buff4,0.0);
//--- 在每个订单号上复制缓冲区价格
   for(int i=0;i<copied;i++)
     {
      //--- 计算缓冲区相应的标引
      int buffer_index=total-copied+i;
      //--- 写下缓冲区的价格
      buff1[buffer_index]=rates[i].open;
      buff2[buffer_index]=rates[i].high;
      buff3[buffer_index]=rates[i].low;
```

© 2000-2025, MetaQuotes Ltd.

```
      buff4[buffer_index]=rates[i].close;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| 随机返回来自市场报价的交易品种                                   |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- 市场报价窗口中显示的交易品种数量
   int symbols=SymbolsTotal(true);
//--- 列表中的交易品种位置 - 从0到交易品种的随机号
   int number=MathRand()%symbols;
//--- 返回指定位置的交易品种名称
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| 改变柱形的外观                                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成柱形属性信息的字符串
   string comm="";
//---改变柱形颜色的模块
   int number=MathRand(); // 获得随机数
//--- 除数等于colors[]数组的大小
   int size=ArraySize(colors);
//--- 获得选择新颜色作为整数除法余数的标引
   int color_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- 写下颜色
   comm=comm+"\r\n"+(string)colors[color_index];
//--- 写下交易品种名称
   comm="\r\n"+symbol+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_LINE

DRAW_COLOR_LINE 值是DRAW_LINE 样式的颜色变量；它也使用指标缓冲区的值绘制线型。但是该样式，类似标题有COLOR单词的所有颜色样式一样，有一个额外特定的指标缓冲区，它可以存储来自特别颜色数组的颜色标引（数量）。因此，每个线型节段的颜色都可以通过指定颜色标引进行定义，在该柱绘制线型。

线的宽度，样式和颜色可以使用编译程序指令 和动态使用 PlotIndexSetInteger() 函数进行设定。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

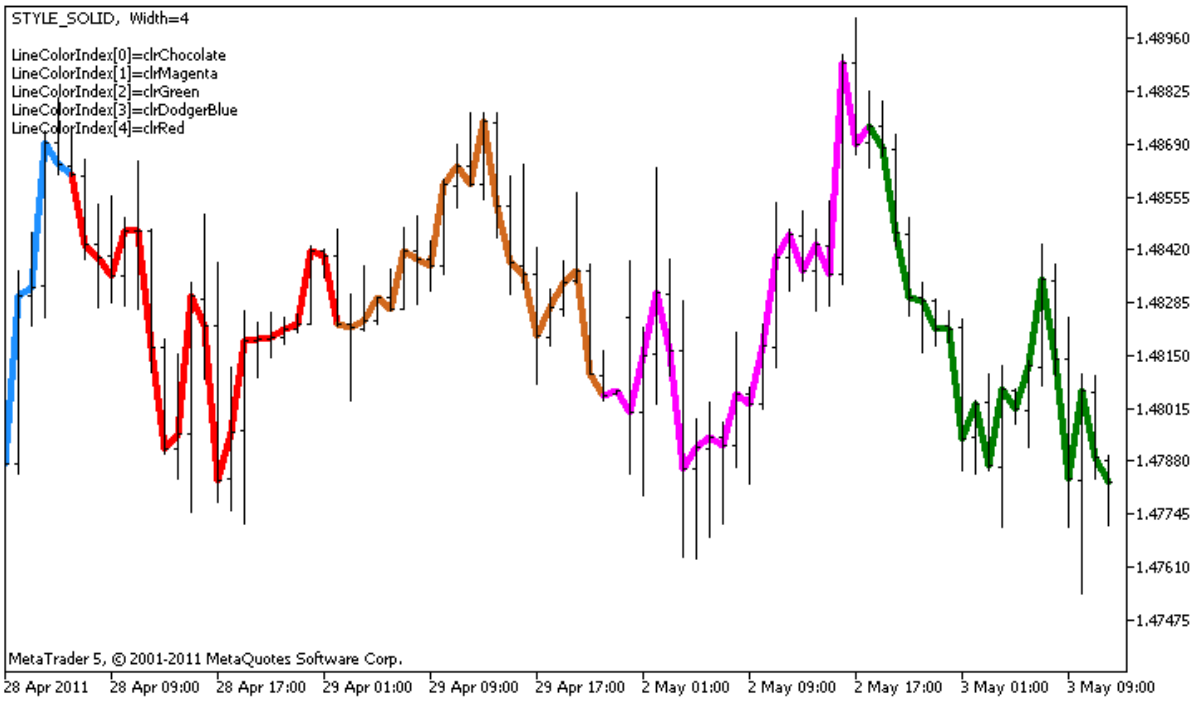
标图DRAW_COLOR_LINE所需的缓冲区数量是 2。

- 一个缓冲区存储用于绘制线型的指标值；
- 一个缓冲区存储每个柱形的线型颜色标引。

颜色可以通过编译程序指令 #property indicator_color1指定，以逗号分隔。颜色数量不能超过64种。

```
//--- 定义5种用于填充每个柱形的颜色（它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 64
```

使用柱形收盘价绘制线型的指标示例。线型的宽度和样式每N=5订单号都会随机变化一次。




线型节段颜色在自定义函数ChangeColors()也会随机变化。

```
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
```

© 2000-2025, MetaQuotes Ltd.

```
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             // 新颜色
      //--- 编写颜色
      comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrinq
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
```

该示例显示了指标"color"版的特点 - 改变线型节段的颜色，您无需改变ColorLineColors[]缓冲区的值（包含颜色标引）。您所需要做的只是在特定数组设置新的颜色。这允许您为整体标图快速改变一次颜色，只使用PlotIndexSetInteger()函数改变小型颜色数组。

请注意，最初DRAW_COLOR_LINE属性的**plot1**使用编译程序指令#property设置，然后在OnCalculate()函数这三种属性随机设置。

N和长度（柱形颜色节段的长度）参数设置在指标的外部参数，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                                DRAW_COLOR_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_LINE"
#property description "It draws a line on Close price in colored pieces of 20 bars eac
#property description "The width, style and color of the line parts are changed randor
#property description "every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图 ColorLine
```

```
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
//--- 定义5种用于填充每个柱形的颜色（它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 64
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int       N=5;            // 改变订单号数量
input int       Length=20;      // 柱形每个颜色部分的长度
int             line_colors=5; // 设置颜色数量是 5 - 见 #property indicator_color1
//--- 标图缓冲区
double          ColorLineBuffer[];
//--- 存储每柱线型颜色的缓冲区
double          ColorLineColors[];

//--- 存储颜色的数组包含7种元素
color colors[]={clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGolde
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 绑定数组和指标缓冲区
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorLineColors,INDICATOR_COLOR_INDEX);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变线型部分的颜色
      ChangeColors(colors,5);
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值的模块
   for(int i=0;i<rates_total;i++)
     {
      //--- 指标值写入缓冲区
      ColorLineBuffer[i]=close[i];
      //--- 现在，为该柱随机设置颜色标引
      int color_index=i%(5*Length);
      color_index=color_index/Length;
      //--- 对于该柱，线型将具有color_index标引的颜色
      ColorLineColors[i]=color_index;
     }

//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //   图形样式数量
                          PLOT_LINE_COLOR,      //   属性标识符
                          plot_color_ind,       //   颜色标引，我们在这里编写颜色
                          cols[i]);             //   新颜色
      //--- 编写颜色
```

© 2000-2025, MetaQuotes Ltd.

```
      comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变指标的线型显示外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+" Width="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   int size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm=EnumToString(styles[style_index])+", "+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_SECTION

DRAW_COLOR_SECTION样式是 DRAW_SECTION的颜色版，但与后者又不太相同，它允许绘制不同颜色的节段。DRAW_COLOR_SECTION样式， 类似标题有 COLOR 单词的所有颜色样式一样，包含一个额外特定的指标缓冲区，它可以存储来自特定颜色数组的颜色标引（数量）。因此，每个节段的颜色都可以通过指定该节段终端对应的柱形标引的颜色标引进行定义。

节段的宽度，颜色和样式的指定类似 DRAW_SECTION 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许 "激活" 指标，以便于可以根据当前状况改变其外观。

从一个缓冲区的非空值到另一个指标缓冲区的非空值绘制节段，空值忽略。若要指定哪个值应被视为 "空"，请在 PLOT_EMPTY_VALUE 属性设定该值：例如，如果指标应该被绘制为非零值的系列节段，然后您需要设置零值为空值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_SECTION,PLOT_EMPTY_VALUE,0);
```

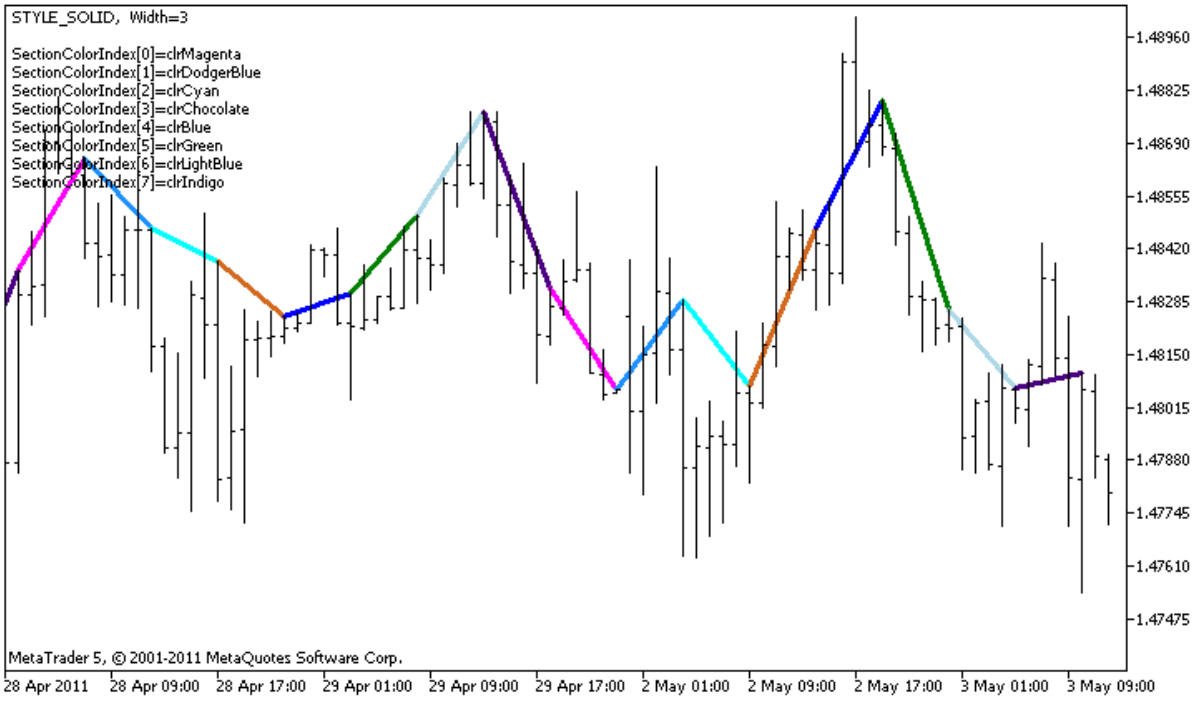
始终明确填写指标缓冲区的值，向不应标图的元素设置缓冲区空值。

标图 DRAW_COLOR_SECTION 所需的缓冲区的数量是2。

- 一个缓冲区存储用于绘制线型的指标值；
- 存储颜色标引的一个缓冲区，它被用于绘制节段（它使得仅设置非空值有意义）。

颜色可以通过编译程序指令 #property indicator_color1指定，以逗号分隔。颜色数量不能超过64种。

使用最高价的值绘制每5柱长的颜色节段的指标示例。节段的颜色，宽度和样式每N 个订单号都会随机变化一次。




© 2000-2025, MetaQuotes Ltd.

请注意，最初DRAW_COLOR_SECTION 8种颜色的 plot1 使用编译程序指令#property设置。然后在OnCalculate()函数，来自颜色数组colors[]的颜色随机设置。

N参数设置在指标的 外部参数 ，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                            DRAW_COLOR_SECTION.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_SECTION"
#property description "It draws colored sections with the length equal to the specifie
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图ColorSection
#property indicator_label1  "ColorSection"
#property indicator_type1   DRAW_COLOR_SECTION
//--- 定义8 种用于填充节段的颜色(它们存储在指定数组)
#property indicator_color1  clrRed,clrGold,clrMediumBlue,clrLime,clrMagenta,clrBrown,c
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      N=5;                        // 改变订单号数量
input int      bars_in_section=5;          // 柱形节段的长度
//--- 计算节段终端的辅助变量
int            divider;
int            color_sections;
//--- 标图缓冲区
double         ColorSectionBuffer[];
//--- 存储每柱线型颜色的缓冲区
double         ColorSectionColors[];
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
```

```
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,ColorSectionBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorSectionColors,INDICATOR_COLOR_INDEX);
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- 为节段填充颜色的颜色数量
   int color_sections=8;   // 请见注释#property indicator_color1
//--- 检查指标参数
   if(bars_in_section<=0)
     {
      PrintFormat("Invalid section length=%d",bars_in_section);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=color_sections*bars_in_section;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式,颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变用于标图节段的颜色
      ChangeColors(colors,color_sections);
      //--- 重置0计数器
      ticks=0;
     }

//--- 指标值计算开始的柱形数
```

© 2000-2025, MetaQuotes Ltd.

```
   int start=0;
//--- 如果之前计算了指标，那么在前一柱开始设置
   if(prev_calculated>0) start=prev_calculated-1;
//--- 这是指标值的全部计算
   for(int i=start;i<rates_total;i++)
     {
      //--- 如果柱数可被section_length整除，它意味着这是节段的终点
      if(i%bars_in_section==0)
        {
         //--- 在该柱最高价设置节段终点
         ColorSectionBuffer[i]=high[i];
         //--- 柱数除以 section_length*number_of_colors的余数
         int rest=i%divider;
         //获得颜色数 =  从 0 到 number_of_colors-1
         int color_indext=rest/bars_in_section;
         ColorSectionColors[i]=color_indext;
        }
      //---如果除法的余数等于柱数，
      else
        {
         //--- 如果什么也没发生，忽略柱形 - 设置0
         else ColorSectionBuffer[i]=0;
        }
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //   图形样式数量
                          PLOT_LINE_COLOR,      //   属性标识符
                          plot_color_ind,       //   颜色标引，我们在这里编写颜色
                          cols[i]);             //   新颜色
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 编写颜色
      comm=comm+StringFormat("SectionColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变指标的线型显示外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+" Width="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   int size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm=EnumToString(styles[style_index])+", "+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_HISTOGRAM

DRAW_COLOR_HISTOGRAM样式根据一系列从0 到指定值的颜色列绘制一个直方图。该值从指标缓冲区获得。每个列都有其自己的来自预设定颜色的颜色。

直方图的宽度，颜色和样式的指定类似 DRAW_HISTOGRAM 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变直方图的外观。

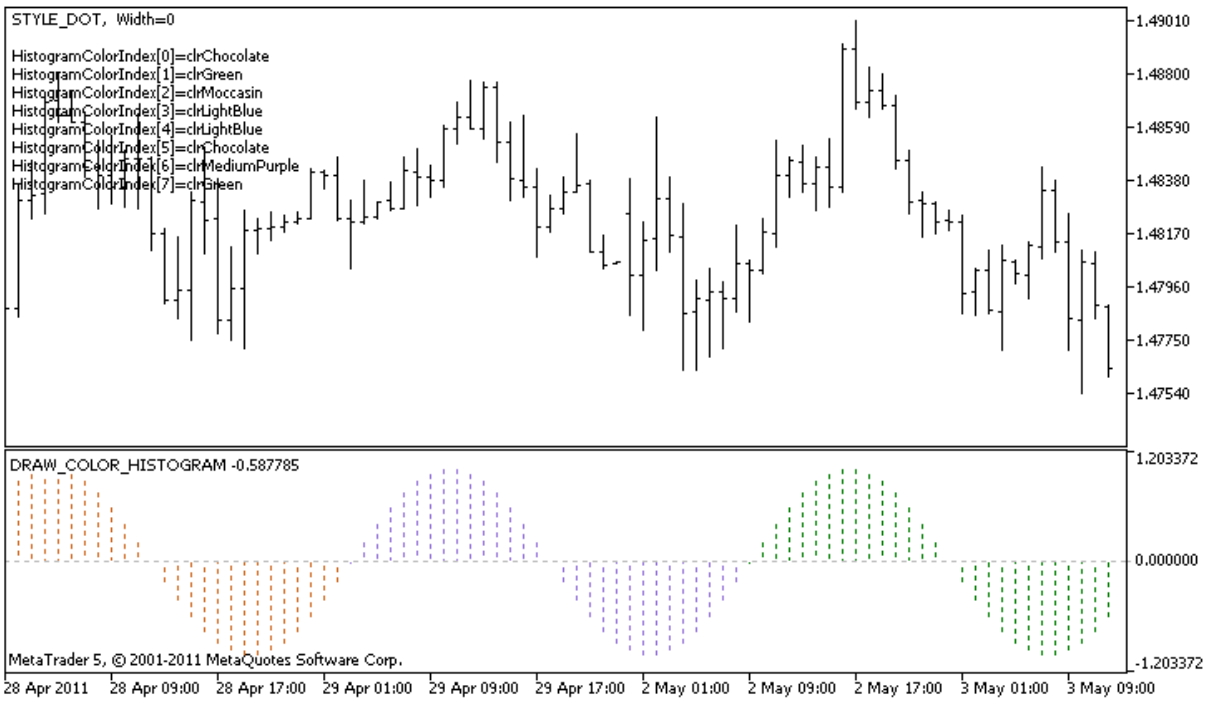
因为每个柱上从0水平绘制列，DRAW_COLOR_HISTOGRAM 应该更好的用在单独的图表窗口。大部分情况下这种类型的标图用于创建oscillator类型的指标，例如， Awesome Oscillator 或 Market Facilitation Index。对于空的不显示的值，应该指定0值。

标图DRAW_COLOR_HISTOGRAM 所需的缓冲区的数量是2。

- 一个缓冲区用于在每柱上存储垂直节段的非零值，节段的第二个终端始终在指标的零线上；
- 存储颜色标引的一个缓冲区，它被用于绘制节段 (它使得仅设置非空值有意义)。

颜色可以使用编译程序指令 #property indicator_color1 指定，以逗号分隔。颜色数量不能超过64种。

基于MathSin()绘制指定颜色正弦曲线的指标示例。所有 直方图列的颜色，宽度和样式每 N个订单号都会随机变化一次。柱形参数指定正弦曲线周期，指定柱形数量之后，正弦曲线将重复这个循环。




请注意，DRAW_COLOR_HISTOGRAM样式的**plot1** ，使用编译程序指令#property设置5个颜色，然后在OnCalculate() 函数随机选择来自存储在colors[] 数组的14种颜色的颜色。N 参数设置在指标的外部参数，使手动配置成为可能 (指标属性窗口的参数标签)。

```
//+------------------------------------------------------------------+
//|                                        DRAW_COLOR_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars to repeat the sinusc

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 输入参数
input int      bars=30;          // 柱形的正弦曲线周期
input int      N=5;              // 改变直方图的订单号数量
//--- 标图 Color_Histogram
#property indicator_label1  "Color_Histogram"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
//--- 定义8 种用于填充节段的颜色(它们存储在指定数组)
#property indicator_color1  clrRed,clrGreen,clrBlue,clrYellow,clrMagenta,clrCyan,clrMe
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 缓冲区的值
double         Color_HistogramBuffer[];
//--- 颜色标引缓冲区
double         Color_HistogramColors[];
//--- 获得2Pi角弧度的因素，当乘以柱的参数时
double         multiplier;
int            color_sections;
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,Color_HistogramBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,Color_HistogramColors,INDICATOR_COLOR_INDEX);
//---- 为正弦曲线填充颜色的颜色数量
   color_sections=8;   //  请见注释#property indicator_color1
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 计算乘数
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Set the value of bars=%d greater than 1",bars);
      //--- 提前终止指标
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变直方图的颜色
      ChangeColors(colors,color_sections);
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值
   int start=0;
//---如果之前OnCalculate开始期间已经计算
   if(prev_calculated>0) start=prev_calculated-1; // 在倒数第二柱设置计算起点
//--- 填写指标缓冲区的值
   for(int i=start;i<rates_total;i++)
     {
      //--- 价值
      Color_HistogramBuffer[i]=sin(i*multiplier);
      //--- 颜色
```

© 2000-2025, MetaQuotes Ltd.

```
      int color_index=i%(bars*color_sections);
      color_index/=bars;
      Color_HistogramColors[i]=color_index;
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             //  新颜色
      //--- 编写颜色
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToS
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变指标的线型显示外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
```

© 2000-2025, MetaQuotes Ltd.

```
   comm=comm+" Width="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   int size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm=EnumToString(styles[style_index])+", "+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_HISTOGRAM2

DRAW_COLOR_HISTOGRAM2 样式使用两个指标缓冲区的值绘制一个指定颜色垂直段的直方图。但与单色DRAW_HISTOGRAM2不同，在该样式中直方图的每一列都有其来自预定义设置的颜色。所有节段值都从指标缓冲区获得。

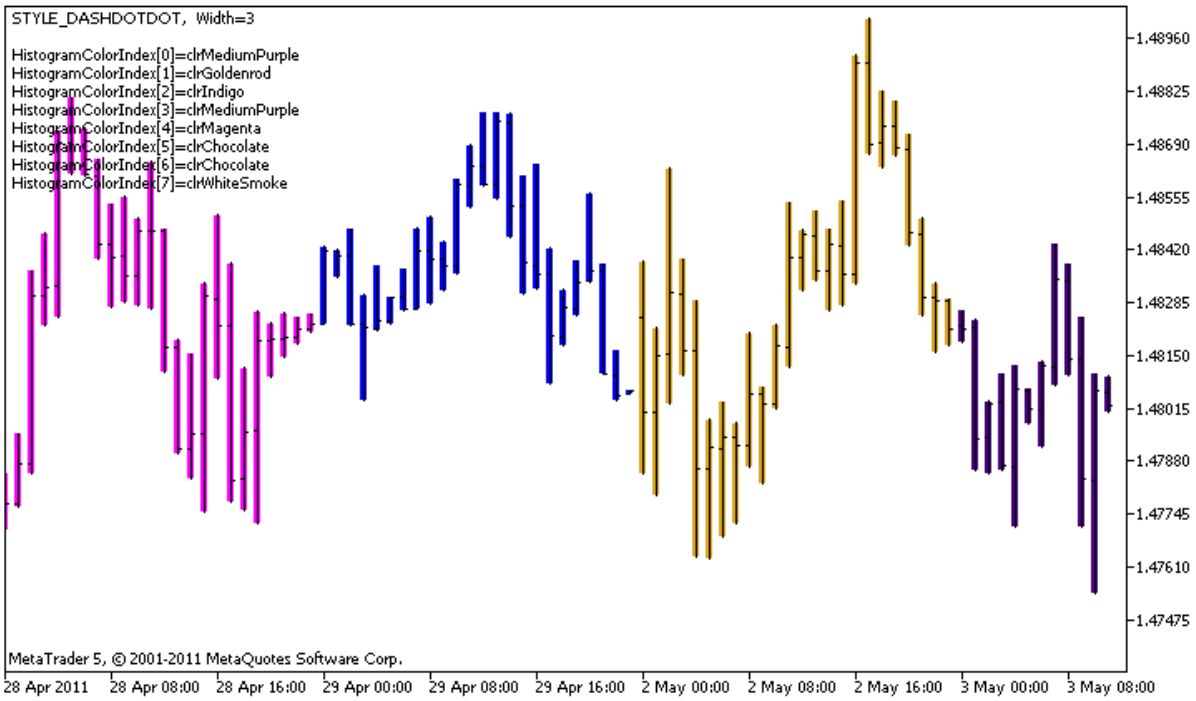
直方图的宽度，样式和颜色的指定类似 DRAW_HISTOGRAM2 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变直方图的外观。

DRAW_COLOR_HISTOGRAM2 样式可以被用在单独的图表子窗口和它的主窗口。由于不绘制空值，指标缓冲区的所有值都需要明确设定。缓冲区不通过空值初始化。

标图DRAW_COLOR_HISTOGRAM2所需的缓冲区的数量是 3：

- 在每个柱上存储垂直段上下端的两个缓冲区；
- 存储颜色标引的一个缓冲区，它被用于绘制节段（它使得仅设置非空值有意义）。

在最高价和最低价之间绘制指定颜色直方图的指标示例。对于每个工作日，直方图的线都有不同的颜色。直方图的日颜色，宽度和样式每 N 个订单号都会随机变化一次。




请注意，DRAW_COLOR_HISTOGRAM2样式的**plot1**，使用编译程序指令#property设置5个颜色，然后在 OnCalculate() 函数随机选择来自存储在colors[] 数组的14种颜色的颜色。

N 参数设置在指标的 外部参数 ，使手动配置成为可能（指标属性窗口的参数标签）。

```
//+------------------------------------------------------------------+
//|                                        DRAW_COLOR_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- 标图ColorHistogram_2
#property indicator_label1  "ColorHistogram_2"
#property indicator_type1   DRAW_COLOR_HISTOGRAM2
//--- 定义 5 种颜色用于为基于工作日的直方图填充颜色（它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- 输入参数
input int       N=5;              // 改变直方图的订单号数量
int             color_sections;
//--- 缓冲区值
double          ColorHistogram_2Buffer1[];
double          ColorHistogram_2Buffer2[];
//--- 颜色标引缓冲区
double          ColorHistogram_2Colors[];
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                                |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,ColorHistogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorHistogram_2Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorHistogram_2Colors,INDICATOR_COLOR_INDEX);
//--- 设置空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- 为正弦曲线填充颜色的颜色数量
   color_sections=8;    // 请见注释#property indicator_color1
//---
   return(INIT_SUCCEEDED);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变用于绘制直方图的颜色
      ChangeColors(colors,color_sections);
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值
   int start=0;
//--- 通过每柱的开盘价获得工作日
   MqlDateTime dt;
//---如果之前OnCalculate开始期间已经计算
   if(prev_calculated>0) start=prev_calculated-1; // 在倒数第二柱设置计算起点
//--- 填写指标缓冲区的值
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
      //--- 价值
      ColorHistogram_2Buffer1[i]=high[i];
      ColorHistogram_2Buffer2[i]=low[i];
      //--- 根据工作日设置颜色标引
      int day=dt.day_of_week;
      ColorHistogram_2Colors[i]=day;
     }
//--- 返回prev_calculated值以便下次调用函数
   return(rates_total);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             //  新颜色
      //--- 编写颜色
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToS
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变指标的线型显示外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+" Width="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   int size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
```

© 2000-2025, MetaQuotes Ltd.

```
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm=EnumToString(styles[style_index])+", "+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_ARROW

DRAW_COLOR_ARROW 样式根据指标缓冲区的值绘制颜色箭头（Wingdings的设置符号）。与DRAW_ARROW相对照，在该样式中为每个符号可以设定来自indicator_color1属性指定的预定义颜色的一种颜色。

交易品种的宽度和颜色的指定类似 DRAW_ARROW 样式 - 使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许基于当前状况改变指标的外观。

交易品种代码使用PLOT_ARROW属性进行设置。

```
//--- 定义PLOT_ARROW绘制的来自Wingdings字体的符号代码到
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

PLOT_ARROW 的默认值=159（一圈）。

每个箭头实际上都是有高度和定位点的符号，可以包含图表上的一些重要信息（例如，柱形的收盘价）。因此，我们可以额外指定像素的垂直移动，这并不依赖图表的比例。箭头沿着指定的像素数量将向下移动，虽然指标的值将会保持相同：

```
//--- 设置箭头的像素垂直移动
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

PLOT_ARROW_SHIFT的负值意味着箭头向上移动，正值表示箭头向下移动。

DRAW_COLOR_ARROW 样式可以被用在单独的图表子窗口和它的主窗口。由于不绘制空值并且也不会出现在"数据窗口"，指标缓冲区的所有值都应该明确设定。缓冲区不通过零值初始化。

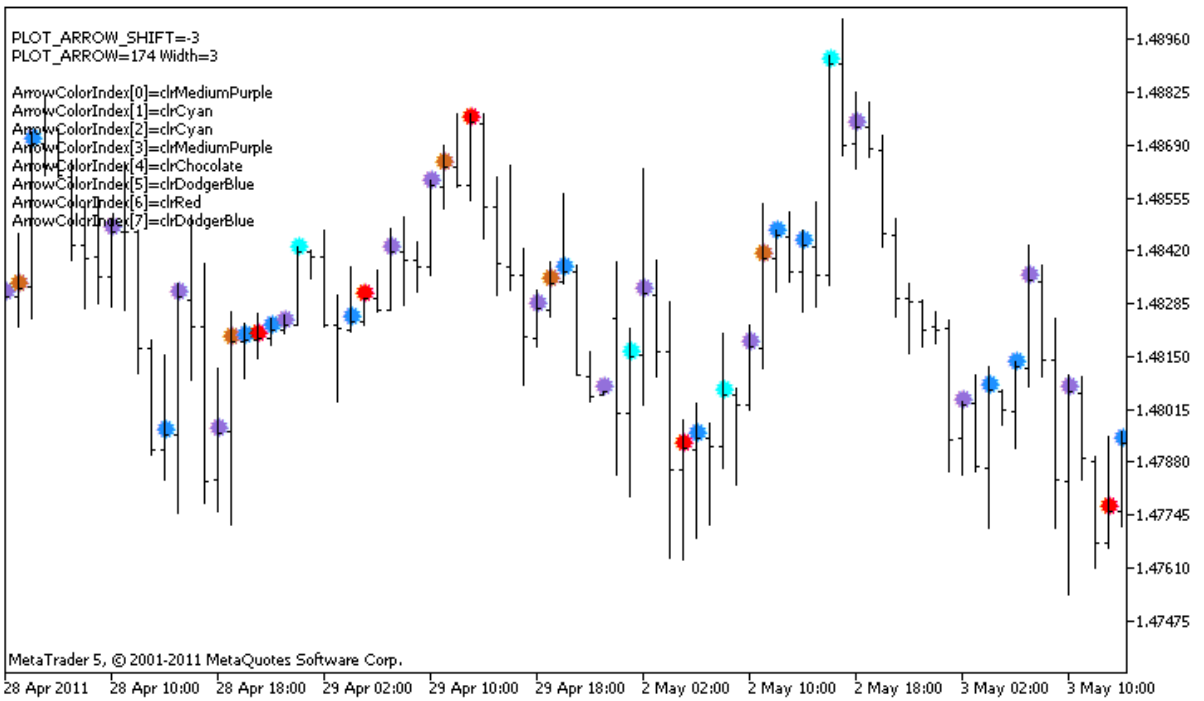
```
//--- Set an empty value
   PlotIndexSetDouble(DRAW_COLOR_ARROW_plot_index,PLOT_EMPTY_VALUE,0);
```

标图DRAW_COLOR_ARROW所需的缓冲区的数量是2。

- 存储价格值的一个缓冲区，它被用于绘制交易品种（加入PLOT_ARROW_SHIFT属性给予的像素移动）；
- 存储颜色标引的一个缓冲区，它被用于绘制箭头（它使得仅设置非空值有意义）。

以比前柱收盘价高的收盘价在每柱上绘制箭头的指标示例。所有 箭头的宽度，移动和符号代码每 N 个订单号都会随机变化一次。交易品种的颜色取决于绘制它的柱数。

© 2000-2025, MetaQuotes Ltd.




在示例中，对于DRAW_COLOR_ARROW样式的**plot1**，属性，颜色和大小使用编译程序指令 #property设置，然后在 OnCalculate() 函数随机设置属性。N 参数设置在指标的 外部参数 ，使手动配置成为可能（指标属性窗口的参数标签）。

请注意，最初的8种颜色使用编译程序指令 #property设置，然后在 OnCalculate()函数，随机设置来自存储在colors[] 数组的14种颜色的颜色。

```
//+------------------------------------------------------------------+
//|                                              DRAW_COLOR_ARROW.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ARROW"
#property description "Draws different-color arrows set by Unicode characters, on a ch
#property description "The color, size, shift and symbol code of the arrow are changed
#property description " randomly every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- 标图ColorArrow
#property indicator_label1  "ColorArrow"
#property indicator_type1   DRAW_COLOR_ARROW
//--- 定义 8 种颜色用于为基于工作日的直方图填充颜色（它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrSeaGreen,clrGold,clrDarkOrange,clrMagent
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- 输入参数
input int      N=5;          // 改变订单号数量
input ushort   code=159;     // 在DRAW_ARROW绘制的符号代码
int            color_sections;
//--- 标图的指标缓冲区
double         ColorArrowBuffer[];
//--- 存储颜色标引的缓冲区
double         ColorArrowColors[];
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,ColorArrowBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorArrowColors,INDICATOR_COLOR_INDEX);
//--- 定义PLOT_ARROW绘制的符号代码
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- 设置箭头的像素垂直移动
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- 设置空值为 0
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- 为正弦曲线填充颜色的颜色数量
   color_sections=8;   //  请见注释#property indicator_color1
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   static int ticks=0;
//--- 计算订单号改变颜色，大小，移动和箭头的代码
   ticks++;
//--- 如果订单号的临界值被积累
   if(ticks>=N)
     {
      //--- 改变箭头属性
      ChangeLineAppearance();
      //--- 改变用于绘制直方图的颜色
      ChangeColors(colors,color_sections);
      //--- 重置0计数器
      ticks=0;
     }

//--- 计算指标值的模块
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- 计算循环
   for(int i=1;i<rates_total;i++)
     {
      //--- 如果当前收盘价高于之前的价格，请绘制一个箭头
      if(close[i]>close[i-1])
         ColorArrowBuffer[i]=close[i];
      //--- 否则指定null 值
      else
         ColorArrowBuffer[i]=0;
      //--- 箭头颜色
      int index=i%color_sections;
      ColorArrowColors[i]=index;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变线段颜色                                                     |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             //  新颜色
      //--- 编写颜色
      comm=comm+StringFormat("ArrowColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrin
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变指标的线型显示外观                                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成线型属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5; // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+" Width="+IntegerToString(width);

//--- 改变箭头代码 (PLOT_ARROW)的模块
   number=MathRand();
//--- 获得整数除法的余数来计算新的箭头代码 从0到19)
   int code_add=number%20;
//--- 设置新的符号代码作为ode+code_add的结果
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- 写下符号代码 PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- 改变箭头的像素垂直移动的模块
   number=MathRand();
//--- 获得整数除法余数的移动
   int shift=20-number%41;
//--- 设置新移动自
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- 写下PLOT_ARROW_SHIFT移动
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- 使用注释在图表上显示信息
   Comment(comm);
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_ZIGZAG

DRAW_COLOR_ZIGZAG 样式使用两个指标缓冲区的值绘制不同颜色的节段。该样式是DRAW_ZIGZAG的颜色版，例如，允许为每个节段指定来自预定义颜色的独立颜色。节段从第一个缓冲区的值标图到第二个指标缓冲区的值。缓冲区不会只包含空值，因为在这种情况下，不会标图。

线的宽度，颜色和样式的指定类似 DRAW_ZIGZAG 样式 - 使用编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许 "激活" 指标，以便于可以根据当前状况改变其外观。

从一个缓冲区的非空值到另一个指标缓冲区的非空值绘制部分。若要指定哪个值应被视为 "空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

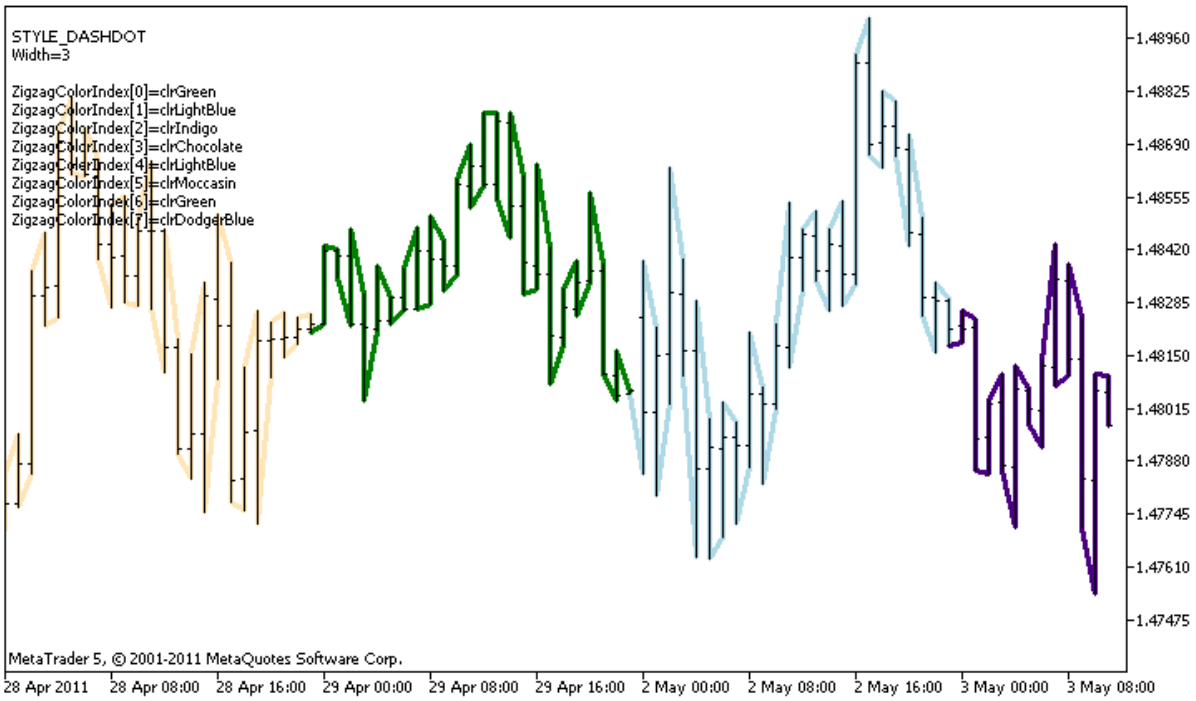
```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_COLOR_ZIGZAG 所需的缓冲区的数量是3：

- 存储zigzag节段终端值的两个缓冲区；
- 存储颜色标引的一个缓冲区，它被用于绘制节段 (它使得仅设置非空值有意义)。

基于最高价和最低价标图的指标示例。zigzag 线的颜色，宽度和样式每N 个订单号都会随机变化一次。




请注意，DRAW_COLOR_ZIGZAG样式的**plot1**，使用编译程序指令#property设置8个颜色，然后在OnCalculate() 函数随机选择来自存储在colors[] 数组的14种颜色的颜色。

N 参数设置在指标的 外部参数 ，使手动配置成为可能 (指标属性窗口的参数标签)。

```
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//|                                                DRAW_COLOR_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ZIGZAG"
#property description "It draws a broken line as a sequence of colored sections, the c
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- 标图Color_Zigzag
#property indicator_label1  "Color_Zigzag"
#property indicator_type1   DRAW_COLOR_ZIGZAG
//--- 定义8 种用于填充节段的颜色(它们存储在指定数组)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int       N=5;              // 改变订单号数量
int             color_sections;
//--- 节段终端缓冲区的值
double          Color_ZigzagBuffer1[];
double          Color_ZigzagBuffer2[];
//--- 节段终端缓冲区的颜色标引
double          Color_ZigzagColors[];
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- 存储线型样式的数组
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                                |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,Color_ZigzagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Color_ZigzagBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Color_ZigzagColors,INDICATOR_COLOR_INDEX);
//----为zigzag填充颜色的颜色数量
```

© 2000-2025, MetaQuotes Ltd.

```
   color_sections=8;   //  请见注释 #property indicator_color1 属性
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和线型宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变用于标图节段的颜色
      ChangeColors(colors,color_sections);
      //--- 重置0计数器
      ticks=0;
     }

//--- 获得每个柱形工作日所需的时间结构
   MqlDateTime dt;

//--- 计算的初始位置
   int start=0;
//--- 如果在前一个订单号计算指标，那么在倒数第二个开始计算
   if(prev_calculated!=0) start=prev_calculated-1;
//--- 计算循环
   for(int i=start;i<rates_total;i++)
     {
      //--- 在结构中写下柱形的开盘时间
      TimeToStruct(time[i],dt);

      //--- 如果柱形数量
      if(i%2==0)
        {
         //---   在第一个缓冲区写下最高价，在第二个缓冲区写下最低价
```

© 2000-2025, MetaQuotes Ltd.

```
         Color_ZigzagBuffer1[i]=high[i];
         Color_ZigzagBuffer2[i]=low[i];
         //--- 节段的颜色
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
      //--- 柱形数量是奇数
      else
        {
         //--- 按相反顺序填写柱形
         Color_ZigzagBuffer1[i]=low[i];
         Color_ZigzagBuffer2[i]=high[i];
         //--- 节段的颜色
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 改变zigzag节段的颜色                                             |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             //  新颜色
      //--- 编写颜色
      comm=comm+StringFormat("ZigzagColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStri
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变zigzag节段的外观                                             |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void ChangeLineAppearance()
  {
//--- 形成Color_ZigZag属性信息的字符串
   string comm="";
//--- 改变线型宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 改变线型样式的模块
   number=MathRand();
//--- 除数等于样式数组的大小
   int size=ArraySize(styles);
//--- 获得选择新样式作为整数除法余数的标引
   int style_index=number%size;
//--- 设置颜色为 PLOT_LINE_COLOR 属性
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- 写下线型样式
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_BARS

DRAW_COLOR_BARS 样式绘制四个指标缓冲区值的柱形，包括开盘价，最高价，最低价和收盘价。该样式是DRAW_BARS 的提高版，允许为每个柱形指定来自预定义颜色的独立颜色。它用于创建自定义指标为柱形，包括单独的图表子窗口中和其他金融工具上的指标。

柱形颜色的设置可以使用 编译程序指令 或动态使用 PlotIndexSetInteger() 函数。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

指标仅向那些柱形绘制，为此，设置所有四个指标缓冲区的非空值。 若要指定哪个值应被视为"空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_BARS,PLOT_EMPTY_VALUE,0);
```

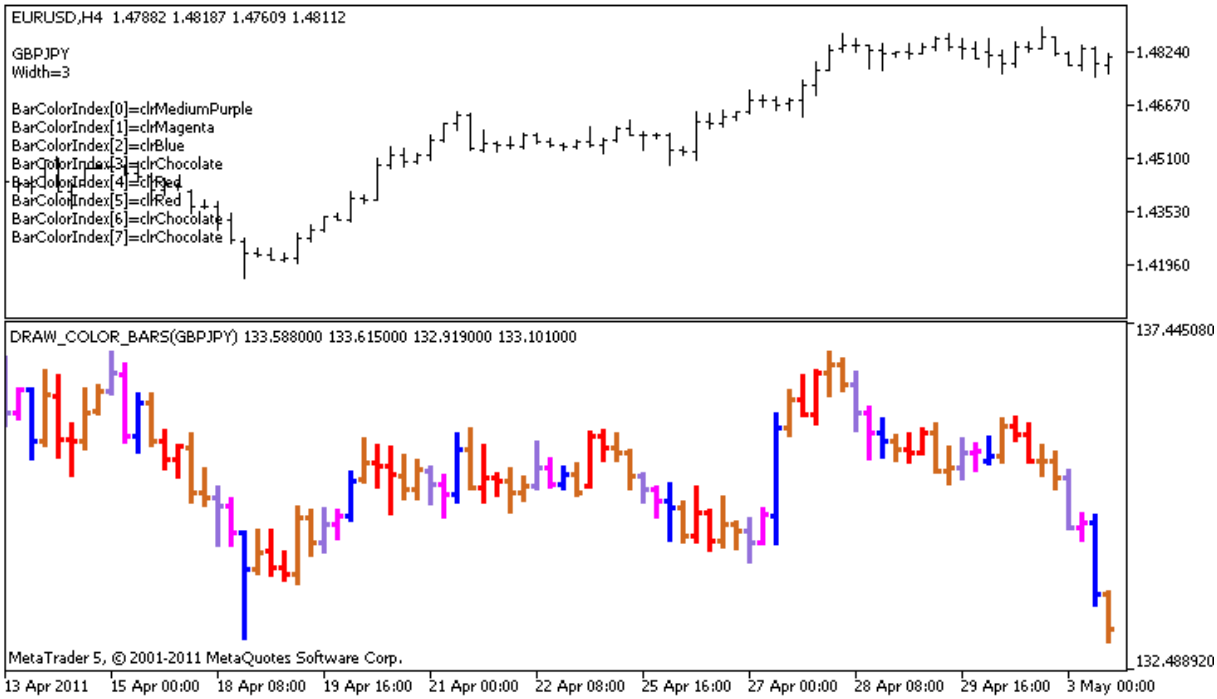
始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_COLOR_BARS 所需的缓冲区的数量是5：

- 存储开盘价，最高价，最低价和收盘价值的四个缓冲区；
- 存储颜色标引的一个缓冲区，它被用于绘制柱形（它使得仅设置将被绘制的柱形有意义）。

标图的所有缓冲区都应该按照给定的顺序一个接一个地进行：开盘价，最高价，最低价，收盘价和颜色缓冲区。价格缓冲区不会只包含nu11值，因为在这种情况下，不会标图。

在单独窗口的选定金融工具上绘制柱形的指标示例。柱形颜色每 N个订单号都会随机变化一次。N 参数设置在指标的 外部参数 ，使手动配置成为可能（指标属性窗口的参数标签）。




请注意，DRAW_COLOR_BARS样式的plot1 ，使用编译程序指令#property设置8个颜色，然后在OnCalculate() 函数随机选择来自存储在colors[] 数组的14种颜色的颜色。

```
//+------------------------------------------------------------------+
//|                                              DRAW_COLOR_BARS.mq5 |
```

© 2000-2025, MetaQuotes Ltd.

```
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_BARS"
#property description "It draws different-color bars of a selected symbol in a separat
#property description "The color and width of bars, as well as the symbol are changed
#property description "every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- 标图 ColorBars
#property indicator_label1  "ColorBars"
#property indicator_type1   DRAW_COLOR_BARS
//--- 定义8种用于填充柱形的颜色 它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参数
input int      N=5;              // 改变类型的订单号数量
input int      bars=500;         // 显示的柱形数量
input bool     messages=false;   // 在"EA交易"日志显示信息
//--- 指标缓冲区
double         ColorBarsBuffer1[];
double         ColorBarsBuffer2[];
double         ColorBarsBuffer3[];
double         ColorBarsBuffer4[];
double         ColorBarsColors[];
//--- 交易品种名称
string symbol;
int    bars_colors;
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,ColorBarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBarsBuffer2,INDICATOR_DATA);
```

```
   SetIndexBuffer(2,ColorBarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorBarsBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorBarsColors,INDICATOR_COLOR_INDEX);
//---- 为柱形填充颜色的颜色数量
   bars_colors=8;   // 请见注释#property indicator_color1 属性
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- 计算订单号改变样式，颜色和柱形宽度
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 选择来自市场报价窗口的新交易品种
      symbol=GetRandomSymbolName();
      //--- 改变线型属性
      ChangeLineAppearance();
      //--- 改变用于绘制蜡烛图的颜色
      ChangeColors(colors,bars_colors);
      int tries=0;
      //--- 试图5 次填充缓冲区的交易品种价格
      while(!CopyFromSymbolToBuffers(symbol,rates_total,bars_colors) && tries<5)
        {
         //--- CopyFromSymbolToBuffers() 函数调用的计数器
         tries++;
        }
      //--- 重置0计数器
      ticks=0;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充指标缓冲区的价格                                             |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total,int bar_colors)
  {
//--- 在rates[] 数组中，我们将复制开盘价，最高价，最低价和收盘价
   MqlRates rates[];
//--- 尝试计数器
   int attempts=0;
//--- 已复制多少
   int copied=0;
//--- 试图25次获得所需交易品种的时间帧
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- 如果复制足够数量的柱形失败
   if(copied!=bars)
     {
      //--- 形成信息字符串
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- 在主图表窗口的注释中显示信息
      Comment(comm);
      //--- 显示信息
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- 设置交易品种的展示
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_BARS("+name+")");
     }
//--- 初始化空值缓冲区
   ArrayInitialize(ColorBarsBuffer1,0.0);
   ArrayInitialize(ColorBarsBuffer2,0.0);
   ArrayInitialize(ColorBarsBuffer3,0.0);
   ArrayInitialize(ColorBarsBuffer4,0.0);

//--- 复制缓冲区的价格
   for(int i=0;i<copied;i++)
     {
      //--- 计算缓冲区相应的标引
      int buffer_index=total-copied+i;
      //--- 写下缓冲区的价格
```

© 2000-2025, MetaQuotes Ltd.

```
      ColorBarsBuffer1[buffer_index]=rates[i].open;
      ColorBarsBuffer2[buffer_index]=rates[i].high;
      ColorBarsBuffer3[buffer_index]=rates[i].low;
      ColorBarsBuffer4[buffer_index]=rates[i].close;
      //---
      ColorBarsColors[buffer_index]=i%bar_colors;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| 随机返回来自市场报价的交易品种                                   |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- 市场报价窗口中显示的交易品种数量
   int symbols=SymbolsTotal(true);
//--- 列表中的交易品种位置 - 从0到交易品种的随机号
   int number=MathRand()%symbols;
//--- 返回指定位置的交易品种名称
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| 改变zigzag节段的颜色                                             |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             // 新颜色
      //--- 编写颜色
      comm=comm+StringFormat("BarColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 改变柱形的外观                                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成柱形属性信息的字符串
   string comm="";

//--- 改变柱形宽度的模块
   int number=MathRand();
//--- 获得整数除法余数的宽度
   int width=number%5;   // 宽度设置从 0 到 4
//--- 设置颜色为PLOT_LINE_WIDTH属性
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- 写下线型宽度
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- 写下交易品种名称
   comm="\r\n"+symbol+comm;

//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DRAW_COLOR_CANDLES

DRAW_COLOR_CANDLES 样式，类似DRAW_CANDLES，使用四个指标缓冲区值绘制蜡烛图，包含开盘价，最高价，最低价和收盘价。另外，它还允许为每个蜡烛图指定一个来自给定设置的颜色。为此，该样式为每个柱形，有一个特定的颜色缓冲区。它用于创建自定义指标为蜡烛图系列，包括单独的图表子窗口中和其他金融工具上的指标。

蜡烛图颜色数量的设置可以使用编译程序指令 或动态使用PlotIndexSetInteger() 函数。动态改变标图属性允许"激活"指标，以便于可以根据当前状况改变其外观。

指标仅向那些柱形绘制，为此，设置四个指标缓冲区价格的非空值。若要指定哪个值应被视为"空"，请在 PLOT_EMPTY_VALUE 属性设定该值：

```
//--- 0(空)值将不参与绘制
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_CANDLES,PLOT_EMPTY_VALUE,0);
```

始终明确填写指标缓冲区的值，在缓冲区设置空值跳过柱形。

标图DRAW_COLOR_CANDLES所需的缓冲区的数量是5：

- 存储开盘价，最高价，最低价和收盘价值的四个缓冲区；
- 存储颜色标引的一个缓冲区，它被用于绘制蜡烛图(它使得仅设置将被绘制的蜡烛图有意义)。

标图的所有缓冲区都应该按照给定的顺序一个接一个地进行：开盘价，最高价，最低价，收盘价和颜色缓冲区。价格缓冲区不会只包含空值，因为在这种情况下，不会标图。

在单独窗口的选定金融工具上绘制蜡烛图的指标示例。蜡烛图颜色每N 个订单号都会随机变化一次。N 参数设置在指标的 外部参数，使手动配置成为可能 (指标属性窗口的参数标签)。

请注意，plot1，使用编译程序指令#property设置颜色，然后在OnCalculate() 函数，从早期已备列表随机设置颜色。

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                            DRAW_COLOR_CANDLES.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate windo
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are
#property description "randomly every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- 标图 ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
//--- 定义8种用于填充蜡烛图的颜色 它们存储在指定数组）
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- 输入参数
input int      N=5;              // 改变类型的订单号数量
input int      bars=500;         // 显示的蜡烛图数量
input bool     messages=false;   // 在"EA交易"日志显示信息
//--- 指标缓冲区
double         ColorCandlesBuffer1[];
double         ColorCandlesBuffer2[];
double         ColorCandlesBuffer3[];
double         ColorCandlesBuffer4[];
double         ColorCandlesColors[];
int            candles_colors;
//--- 交易品种名称
string symbol;
//--- 存储颜色的数组包含14种元素
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 如果柱形非常小 - 提前完成工作
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- 指标缓冲区映射
   SetIndexBuffer(0,ColorCandlesBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorCandlesBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorCandlesBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorCandlesBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- 空值
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- 绘制柱形的交易品种名称
   symbol=_Symbol;
//--- 设置交易品种的展示
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
//---- 为蜡烛图填充颜色的颜色数量
   candles_colors=8;      //  请见注释#property indicator_color1 属性
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=INT_MAX-100;
//--- 计算订单号改变样式，颜色
   ticks++;
//--- 如果足够数量的订单号被积累
   if(ticks>=N)
     {
      //--- 选择来自市场报价窗口的新交易品种
      symbol=GetRandomSymbolName();
      //--- 改变格式
      ChangeLineAppearance();
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 改变用于绘制蜡烛图的颜色
      ChangeColors(colors,candles_colors);

      int tries=0;
      //--- 试图5 次填充缓冲区plot1的交易品种价格
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            ColorCandlesBuffer1,ColorCandlesBuffer2,ColorCandlesBuffer3,
            ColorCandlesBuffer4,ColorCandlesColors,candles_colors)
            && tries<5)
        {
         //--- CopyFromSymbolToBuffers() 函数调用的计数器
         tries++;
        }
      //--- 重置0计数器
      ticks=0;
     }
//--- 返回 prev_calculated值以便下次调用函数
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充指定蜡烛图                                                   |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[],
                             double &col_buffer[],
                             int    cndl_colors
                             )
  {
//--- 在rates[] 数组中，我们将复制开盘价，最高价，最低价和收盘价
   MqlRates rates[];
//--- 尝试计数器
   int attempts=0;
//--- 已复制多少
   int copied=0;
//--- 试图25次获得所需交易品种的时间帧
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- 如果复制足够数量的柱形失败
   if(copied!=bars)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 形成信息字符串
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- 在主图表窗口的注释中显示信息
      Comment(comm);
      //--- 显示信息
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- 设置交易品种的展示
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
     }
//--- 初始化空值缓冲区
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
   ArrayInitialize(buff4,0.0);
//--- 在每个订单号上复制缓冲区价格
   for(int i=0;i<copied;i++)
     {
      //--- 计算缓冲区相应的标引
      int buffer_index=total-copied+i;
      //--- 写下缓冲区的价格
      buff1[buffer_index]=rates[i].open;
      buff2[buffer_index]=rates[i].high;
      buff3[buffer_index]=rates[i].low;
      buff4[buffer_index]=rates[i].close;
      //--- 设置蜡烛图颜色
      int color_index=i%cndl_colors;
      col_buffer[buffer_index]=color_index;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| 随机返回来自市场报价的交易品种                                   |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- 市场报价窗口中显示的交易品种数量
   int symbols=SymbolsTotal(true);
//--- 列表中的交易品种位置 - 从0到交易品种的随机号
   int number=MathRand()%symbols;
//--- 返回指定位置的交易品种名称
```

© 2000-2025, MetaQuotes Ltd.

```
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| 改变蜡烛图节段的颜色                                             |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- 颜色数
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- 为每个颜色标引随机定义一个新的颜色
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- 获得随机数
      int number=MathRand();
      //--- 获得col[]数组的标引作为整数除法的余数
      int i=number%size;
      //--- 设置每个标引的颜色为 PLOT_LINE_COLOR属性
      PlotIndexSetInteger(0,                    //  图形样式数量
                          PLOT_LINE_COLOR,      //  属性标识符
                          plot_color_ind,       //  颜色标引，我们在这里编写颜色
                          cols[i]);             //  新颜色
      //--- 编写颜色
      comm=comm+StringFormat("CandleColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStri
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| 改变蜡烛图的外观                                                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- 形成蜡烛图属性信息的字符串
   string comm="";
//--- 写下交易品种名称
   comm="\r\n"+symbol+comm;
//--- 使用注释在图表上显示信息
   Comment(comm);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 指标属性和函数间的连接

每个自定义指标都有很多属性，有些是强制的，有些是开始描述的仓位，具有如下属性：

- 窗口指示到绘图指示 -indicator_separate_window 或者 indicator_chart_window；
- 指标缓冲区数量 -indicator_buffers；
- 指标绘图数量 -indicator_plots。

也可以通过 预处理器 指令或者建立自定义指标的函数建立其他属性。这些属性和类似函数在如下表格中描述。

| 指标子窗口属性指令 | IndicatorSet...()型函数 | 调整过的子窗口属性指令 |
|---|---|---|
| indicator_height | IndicatorSetInteger (INDICATOR_INDICATOR_HEIGHT, nHeight) | 子窗口高度的修正值 |
| indicator_minimum | IndicatorSetDouble (INDICATOR_MINIMUM, dMaxValue) | 纵轴最小值 |
| indicator_maximum | IndicatorSetDouble (INDICATOR_MAXIMUM, dMinValue) | 纵轴最大值 |
| indicator_levelN | IndicatorSetDouble (INDICATOR_LEVELVALUE, N-1, nLevelValue) | N水平面纵轴值 |
| no preprocessor directive | IndicatorSetString (INDICATOR_LEVELTEXT, N-1, sLevelName) | 显示的水平面名称 |
| indicator_levelcolor | IndicatorSetInteger (INDICATOR_LEVELCOLOR, N-1, nLevelColor) | N水平面颜色 |
| indicator_levelwidth | IndicatorSetInteger (INDICATOR_LEVELWIDTH, N-1, nLevelWidth) | N平面的线宽 |
| indicator_levelstyle | IndicatorSetInteger (INDICATOR_LEVELSTYLE, N-1, nLevelStyle) | N平面的线型 |
| 绘画属性指令 | PlotIndexSet...()型函数 | 调整过的绘画属性指令 |
| indicator_labelN | PlotIndexSetString(N-1,PLOT_LABEL,sLabel) | 图N缩略名。当鼠标光标放在上面时显示在数据窗口和弹出提示框中 |
| indicator_colorN | PlotIndexSetInteger(N-1, PLOT_LINE_COLOR, nColor) | 图N线的颜色 |
| indicator_styleN | PlotIndexSetInteger(N-1, PLOT_LINE_STYLE, nType) | 图N的线型 |

© 2000-2025, MetaQuotes Ltd.

| 指标子窗口属性指令 | IndicatorSet...()型函数 | 调整过的子窗口属性指令 |
|---|---|---|
| indicator_typeN | PlotIndexSetInteger(N-1, PLOT_DRAW_TYPE, nType) | 图N的线种 |
| indicator_widthN | PlotIndexSetInteger(N-1, PLOT_LINE_WIDTH, nWidth) | 图N的线宽 |
| **常用指标属性** | **IndicatorSet...()型函数** | **描述** |
| 无预处理器指令 | IndicatorSetString (INDICATOR_SHORTNAME, sShortName) | 设置显示在指标列表（在程序端按**Ctrl+I**打开）中的指标的便捷短名称。 |
| 无预处理器指令 | IndicatorSetInteger (INDICATOR_DIGITS, nDigits) | 设置指标值显示所需精确性-小数位的数量。 |
| 无预处理器指令 | IndicatorSetInteger (INDICATOR_LEVELS, nLevels) | 设置指标窗口平面数 |
| indicator_applied_price | 无函数，属性仅可以通过预处理器指令设置。 | 用于指标计算的默认价格类型。只有使用OnCalculate()时被指定。<br>属性值也可以在“参数”标签中“应用到”指标属性对话框中设置。 |

记录水平计算和预处理器项目绘图以1开始，而相同属性的技术使用函数以0开始，例如，少于1的指示值为#property。

有许多指令，没有类似函数：

| 指令 | 描述 |
|---|---|
| indicator_chart_window | 指标显示在主窗口中 |
| indicator_separate_window | 指标显示在独立子窗口中 |
| indicator_buffers | 所需指标缓冲区的数量 |
| indicator_plots | 指标中的平面图数量 |

© 2000-2025, MetaQuotes Ltd.

# SetIndexBuffer

该函数在全局水平使用一维动态双精度数组捆绑指定指标缓冲区。

```
bool  SetIndexBuffer(
   int                     index,          // 缓冲指数
   double                  buffer[],       // 数组
   ENUM_INDEXBUFFER_TYPE   data_type       // 要存储的东西
   );
```

参量

*index*

[in]指标缓冲区的数字，编号从0开始，该数字在#property indicator_buffers中减少申报价格。

*buffer[]*

[in] 在自定义指标程序中表明一组数组。

*data_type*

[in] 数据存储类型在指标数组里。默认是 INDICATOR_DATA （计算指标的值），也可以使用 INDICATOR_COLOR_INDEX 值；在这种情况下缓冲区为先前指标缓冲区存储颜色指数。可以在 #property indicator_colorN 水平上指定64种颜色, INDICATOR_CALCULATIONS 值表示缓冲器在指标媒介计算中使用，而不是用来绘画的。

返回值

如果成功，返回 true，否则 -false。

注释

在捆绑后，动态数组buffer[]将在共同数组中编入索引，尽管 时间序列 索引已经在界限数组中预先安装，如果想要转变指标数组元素的接入命令，在使用SetIndexBuffer()函数后捆绑数组使用 ArraySetAsSeries()函数。注释表明通过SetIndexBuffer()。 函数不能改变建立指标缓冲区中的动态数组的大小，对于指标缓冲器来说，所有大小转变的操作都可以通过程序的子系统来执行。

示例：

```
//+------------------------------------------------------------------+
//|                                                TestCopyBuffer1.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 图 MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参量
input bool                AsSeries=true;
input int                 period=15;
input ENUM_MA_METHOD      smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE  price=PRICE_CLOSE;
input int                 shift=0;
//--- 指标缓冲
double                    MABuffer[];
int                       ma_handle;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲绘图
   if(AsSeries) ArraySetAsSeries(MABuffer,true);
   Print("Indicator buffer is timeseries = ",ArrayGetAsSeries(MABuffer));
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Indicator buffer after SetIndexBuffer() is timeseries = ",
         ArrayGetAsSeries(MABuffer));

//--- 更待指标缓冲访问单元命令
   ArraySetAsSeries(MABuffer,AsSeries);

   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 复制缓冲区MABuffer中的移动平均值
   int copied=CopyBuffer(ma_handle,0,0,rates_total,MABuffer);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("MABuffer[0] = ",MABuffer[0]);// 取决于AsSeries值
                                       // 将收到旧值
                                       // 或者为当前未完成的字节

//--- 为下一次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

另见

自定义指标属性, 访问时间序列和指标

© 2000-2025, MetaQuotes Ltd.

# IndicatorSetDouble

函数建立了类似指标属性值，指标属性应该是双精度类型，有2个变量函数可以使用。

调用属性的指示标识符。

```
bool  IndicatorSetDouble(
   int      prop_id,           // 标识符
   double   prop_value         // 将被设置的值
   );
```

调用指示标识符和属性修饰语。

```
bool  IndicatorSetDouble(
   int      prop_id,           // 标识符
   int      prop_modifier,     // 修饰符
   double   prop_value         // 将被设置的值
   )
```

参量

*prop_id*

[in]值可以是 ENUM_CUSTOMIND_PROPERTY_DOUBLE 值中的一个。

*prop_modifier*

[in] 规定属性的修饰语，只有水平指示需要修饰语。水平编号从0开始。它表示若要为第二水平设置属性您需要指定1 (少于使用编译程序指令时)。

*prop_value*

[in]属性值。

返回值

如果成功，返回 true，否则 -false。

注意

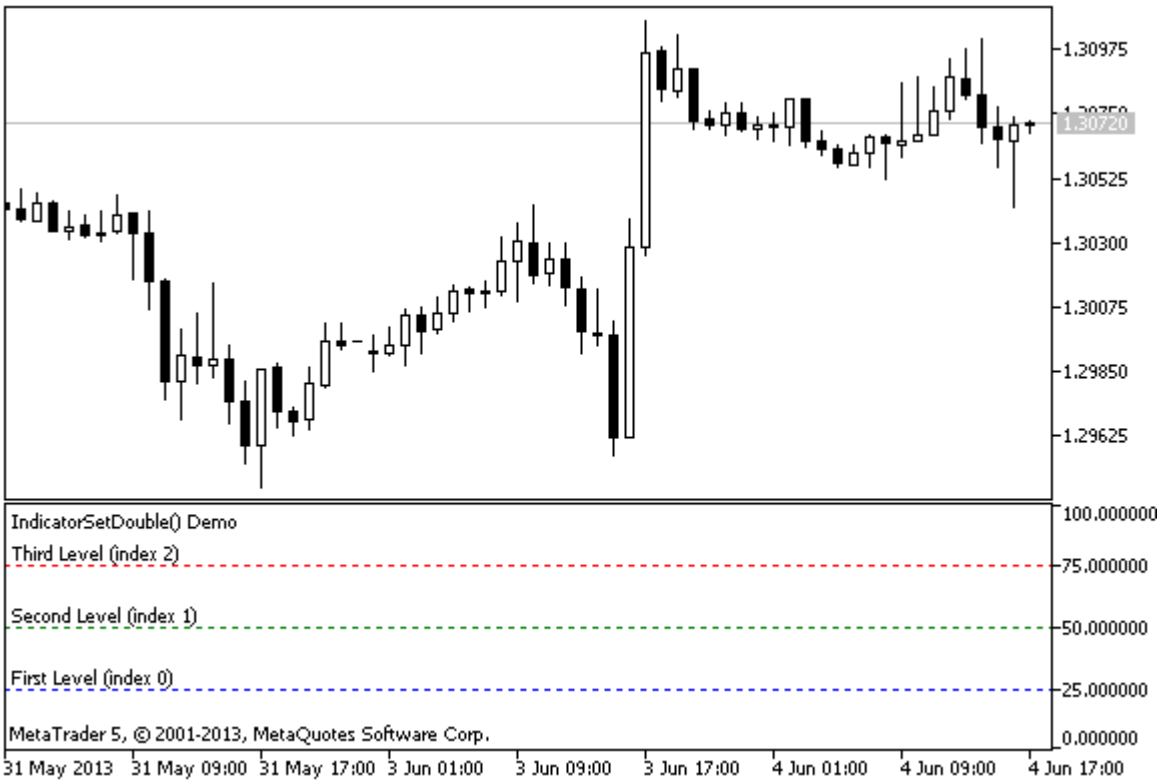
使用#property指令时属性 (修饰符)的编号从1 (一)开始，而函数使用从0 (零)开始编号。如果水平编号设置错误，那么指标展示可能会不同于预想。

例如，在单独子窗口的指标第一水平值可以通过两种方式设置：

- property indicator_level1 50 - 值1用于指定水平编号，
- IndicatorSetDouble(INDICATOR_LEVELVALUE, 0, 50) - 0 用于指定第一水平。

例如：颠倒指标窗口最大最小值和水平线所在水平值的指标。

© 2000-2025, MetaQuotes Ltd.




```
#property indicator_separate_window
//--- 设置指标窗口的最大最小值
#property indicator_minimum  0
#property indicator_maximum  100
//--- 在单独指标窗口展示三条水平线
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
//--- 设置水平线的粗细
#property indicator_levelwidth 1
//--- 设置水平线的样式
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 设置水平线的描述
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- 设置指标缩略名
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- 设置每个水平的颜色
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,clrGreen);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,2,clrRed);
```

© 2000-2025, MetaQuotes Ltd.

```
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
   static double level1=25,level2=50,level3=75;
   static double max=100,min=0, shift=100;
//--- 计算订单号
   tick_counter++;
//--- 每10个订单号颠倒一次水平
   if(tick_counter%10==0)
     {
      //--- 水平值的颠倒符号
      level1=-level1;
      level2=-level2;
      level3=-level3;
      //--- 最大最小值的颠倒符号
      max-=shift;
      min-=shift;
      //--- 颠倒替换值
      shift=-shift;
      //--- 设置新的水平值
      IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,2,level3);
      //--- 在指标窗口设置新的最大最小值
      Print("Set up max = ",max,",   min = ",min);
      IndicatorSetDouble(INDICATOR_MAXIMUM,max);
      IndicatorSetDouble(INDICATOR_MINIMUM,min);
     }
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
```

另见

© 2000-2025, MetaQuotes Ltd.

指标类型示例, 指标属性和函数间的连接, 绘画风格

© 2000-2025, MetaQuotes Ltd.

# IndicatorSetInteger

函数建立类似指标属性值，属性必须是整型或者颜色型的，有2个变量函数可以使用。

调用属性的指示标识符。

```
bool  IndicatorSetInteger(
   int  prop_id,           // 标识符
   int  prop_value         // 将被设置的值
   );
```

调用指示标识符和属性修饰语。

```
bool  IndicatorSetInteger(
   int  prop_id,           // 标识符
   int  prop_modifier,     // 修饰符
   int  prop_value         // 将被设置的值
   )
```

参量

*prop_id*

[in] 值可以是 ENUM_CUSTOMIND_PROPERTY_INTEGER 值中一个。

*prop_modifier*

[in] 指定属性修饰语，值有水平属性需要修饰语。

*prop_value*

[in] 属性值。

返回值

如果成功，返回 true，否则 -false.

注意

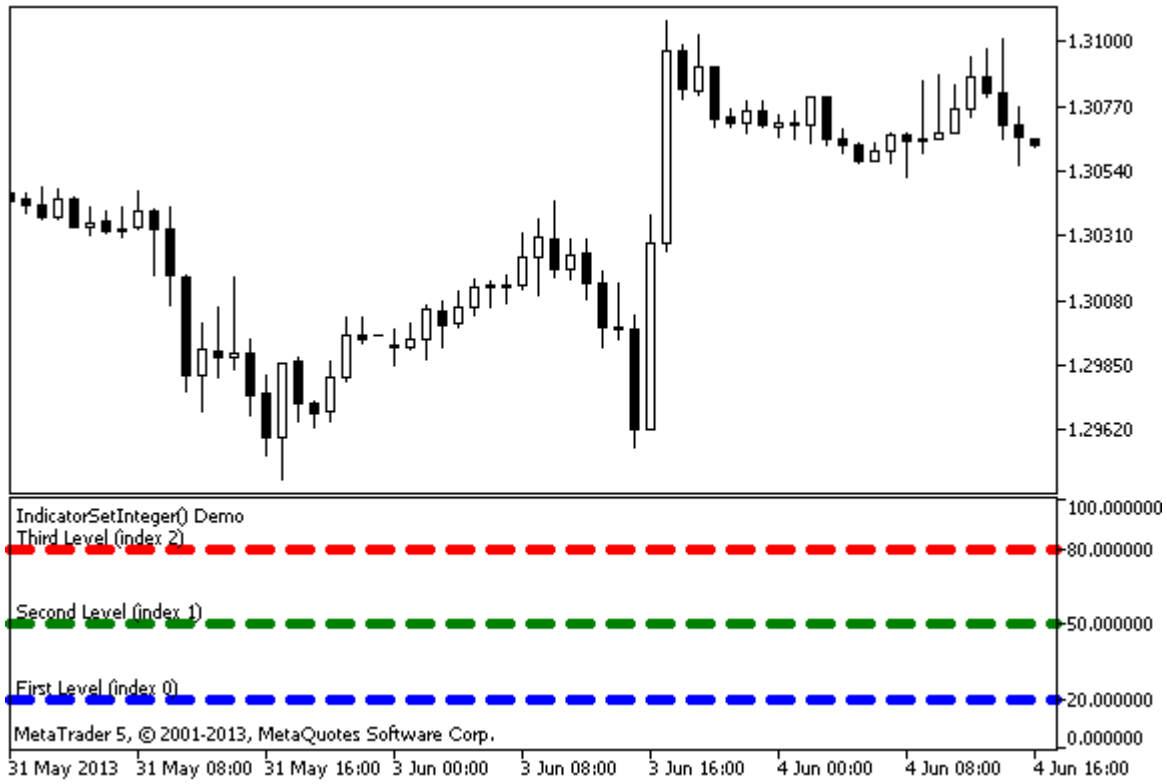
使用#property指令时属性（修饰符）的编号从1（一）开始，而函数使用从0（零）开始编号。如果水平编号设置错误，那么 指标展示 可能会不同于预想。

例如，若要设置第一水平线的粗细，请使用0标引：

- IndicatorSetInteger(INDICATOR_LEVELWIDTH, **0**, 5) -标引 0 用于设定第一水平的粗细。

例如：设置指标水平线的颜色，样式和粗细的指标。

© 2000-2025, MetaQuotes Ltd.




```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- 在单独指标窗口展示三条水平线
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- 设置水平线的粗细
#property indicator_levelwidth 5
//--- 设置水平线的颜色
#property indicator_levelcolor clrAliceBlue
//--- 设置水平线的样式
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 设置水平线的描述
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- 设置指标缩略名
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetInteger() Demo");
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
//--- 计算订单号
   tick_counter++;
//--- 根据订单号计数器计算水平线的颜色
   ChangeLevelColor(0,tick_counter,3,6,10); // 三个最后的参数切换颜色
   ChangeLevelColor(1,tick_counter,3,6,8);
   ChangeLevelColor(2,tick_counter,4,7,9);
//--- 更改水平线的样式
   ChangeLevelStyle(0,tick_counter);
   ChangeLevelStyle(1,tick_counter+5);
   ChangeLevelStyle(2,tick_counter+15);
//--- 获得宽度为订单号整数除法除以5的余数
   int width=tick_counter%5;
//--- 迭代所有水平线并设置粗细
   for(int l=0;l<3;l++)
      IndicatorSetInteger(INDICATOR_LEVELWIDTH,l,width+1);
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 在单独指标窗口设置水平线颜色                                      |
//+------------------------------------------------------------------+
void ChangeLevelColor(int level,       // 水平线数
                      int tick_number,// 被除数，获得除法余数的数
                      int f_trigger,   // 第一个颜色切换除数
                      int s_trigger,   // 第二个颜色切换除数
                      int t_trigger)   // 第三个颜色切换除数
  {
   static color colors[3]={clrRed,clrBlue,clrGreen};
//--- colors[]数组的颜色标引
   int index=-1;
//--- 计算来自colors[]数组的颜色数为水平线涂色
   if(tick_number%f_trigger==0)
      index=0;    // 如果tick_number被f_trigger除，没有余数
   if(tick_number%s_trigger==0)
      index=1;    // 如果tick_number被s_trigger除，没有余数
```

© 2000-2025, MetaQuotes Ltd.

```
   if(tick_number%t_trigger==0)
      index=2;    // 如果tick_number被t_trigger除，没有余数
//--- 如果定义颜色，设置它
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELCOLOR,level,colors[index]);
//---
  }
//+------------------------------------------------------------------+
//| 在单独指标窗口设置水平线的样式                                   |
//+------------------------------------------------------------------+
void ChangeLevelStyle(int level,     // 水平线数
                      int tick_number// 获得除法余数的数
                      )
  {
//--- 存储样式的数组
   static ENUM_LINE_STYLE styles[5]=
     {STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- styles[]数组的样式标引
   int index=-1;
//--- 计算styles[]数组的数字来设置水平线的样式
   if(tick_number%50==0)
      index=5;    // 如果tick_number 被50除没有余数，那么样式是STYLE_DASHDOTDOT
   if(tick_number%40==0)
      index=4;    // ... 样式是 STYLE_DASHDOT
   if(tick_number%30==0)
      index=3;    // ... STYLE_DOT
   if(tick_number%20==0)
      index=2;    // ... STYLE_DASH
   if(tick_number%10==0)
      index=1;    // ... STYLE_SOLID
//--- 如果定义样式，设置它
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELSTYLE,level,styles[index]);
  }
```

**See also**

Custom Indicator Properties, Program Properties (#property), Drawing Styles

© 2000-2025, MetaQuotes Ltd.

# IndicatorSetString

函数建立类似指标属性值，指标属性必须是字符串型。有2个变量函数可以使用。

调用属性的指示标识符

```
bool  IndicatorSetString(
   int     prop_id,           // 标识符
   string  prop_value         // 将被设置的值
   );
```

调用指示标识符和属性修饰语

```
bool  IndicatorSetString(
   int     prop_id,           // 标识符
   int     prop_modifier,     // 修饰符
   string  prop_value         // 将被设置的值
   )
```

参量

*prop_id*

[in] 值可以是 ENUM_CUSTOMIND_PROPERTY_STRING 值中一个。

*prop_modifier*

[in] 指定属性标识符，只有水平属性需要修饰语。

*prop_value*

[in] 属性值。

返回值

如果成功，返回 true，否则 -false。

注意

使用#property指令时属性（修饰符）的编号从1（一）开始，而函数使用从0（零）开始编号。如果水平编号设置错误，那么指标展示可能会不同于预想。

例如，若要设置第一水平线的描述，请使用0标引：

- IndicatorSetString(INDICATOR_LEVELTEXT, **0**, "First Level") - 标引 0用于设定第一水平的文本描述。

例如：设置指标水平线的文本标签的指标。

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- 在单独指标窗口展示三条水平线
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- 设置水平线的颜色
#property indicator_levelcolor clrRed
//--- 设置水平线的样式
#property indicator_levelstyle STYLE_SOLID
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 设置水平线的描述
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- 设置指标缩略名
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetString() Demo");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---

//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
```

另见

自定义指标属性, 程序属性 (#property)

© 2000-2025, MetaQuotes Ltd.

# PlotIndexSetDouble

该函数建立一定特征的相应值，指标性质应该是双精度型。

```
bool  PlotIndexSetDouble(
   int     plot_index,      // 图样式指数
   int     prop_id,         // 属性标识符
   double  prop_value       // 将被设置的值
   );
```

参量

*plot_index*

[in] 图示 的索引

*prop_id*

[in] 该值可以是 ENUM_PLOT_PROPERTY_DOUBLE 值中的一个。

*prop_value*

[in] 属性值。

返回值

如果成功，返回 true，否则 false。

© 2000-2025, MetaQuotes Ltd.

# PlotIndexSetInteger

该函数建立一定特征的相应值，指标性质应该是整型，图表型，布尔型或者颜色型。有两种变量函数。

调用指定的属性标识符。

```
bool  PlotIndexSetInteger(
   int  plot_index,         // 图样式指数
   int  prop_id,            // 属性标识符
   int  prop_value          // 将被设置的值
   );
```

调用指定的属性标识符和修饰符。

```
bool  PlotIndexSetInteger(
   int  plot_index,         // 图样式指数
   int  prop_id,            // 属性标识符
   int  prop_modifier,      // 属性修饰符
   int  prop_value          // 将被设置的值
   )
```

参量

*plot_index*

[in] 图示 的索引

*prop_id*

[in] 该值可以是 ENUM_PLOT_PROPERTY_INTEGER 值中的一个。

*prop_modifier*

[in] 规定属性的修饰符，只有显色指数性能要求修改。

*prop_value*

[in] 属性值。

返回值

如果成功，返回 true，否则 false。

示例：这是画有三色线的指标。每五个节点改变一次颜色。

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//---- 图线颜色
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrGreen,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- 指标缓冲区
double          ColorLineBuffer[];
double          ColorBuffer[];
int             MA_handle;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲绘图
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);
//--- 获得MA处理器
   MA_handle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| 获得颜色指数                                                     |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int getIndexOfColor(int i)
  {
   int j=i%300;
   if(j<100) return(0);// 第一指数
   if(j<200) return(1);// 第二指数
   return(2); // 第三指数
  }
//+------------------------------------------------------------------+
//| 自定义指标反复函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   static int ticks=0,modified=0;
   int limit;
//--- 第一计算或者柱数被改变
   if(prev_calculated==0)
     {
      //--- 复制MA值到指标缓冲区ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);// 复制失败-丢弃
      //--- 现在为每个柱设置线的颜色
      for(int i=0;i<rates_total;i++)
         ColorBuffer[i]=getIndexOfColor(i);
     }
   else
     {
      //--- 复制MA值到指标缓冲区ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);

      ticks++;// 数订单号
      if(ticks>=5)//改变配色方案的时候
        {
         ticks=0; // 复位计数器
         modified++; // 颜色更改的计数器
         if(modified>=3)modified=0;// 复位计数器
         ResetLastError();
         switch(modified)
           {
```

© 2000-2025, MetaQuotes Ltd.

```
            case 0:// 第一配色方案
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrRed);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrBlue);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrGreen);
               Print("Color scheme "+modified);
               break;
            case 1:// 第二配色方案
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrYellow);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrPink);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLightSlateGray);
               Print("Color scheme "+modified);
               break;
            default:// 第三配色方案
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrLightGoldenrod);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrOrchid);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLimeGreen);
               Print("Color scheme "+modified);
           }
        }
      else
        {
         //--- 设置初始位置
         limit=prev_calculated-1;
         //--- 现在为每柱设置线的颜色
         for(int i=limit;i<rates_total;i++)
            ColorBuffer[i]=getIndexOfColor(i);
        }
     }
//--- 为下一次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# PlotIndexSetString

该函数设置了相应指标线的相应属性值。指标属性必须是一种字符串类型。

```
bool  PlotIndexSetString(
   int      plot_index,      // 图样式指数
   int      prop_id,         // 属性标识符
   string   prop_value       // 将被设置的值
   );
```

参量

*plot_index*

[in] 图示索引

*prop_id*

[in] 该值可以是ENUM_PLOT_PROPERTY_STRING 列举值中的一个。

*prop_value*

[in] 属性值。

返回值

如果成功，返回 true，否则 false。

© 2000-2025, MetaQuotes Ltd.

# PlotIndexGetInteger

该函数建立一定特征的相应值，指标性质应该是整型，颜色型，布尔型或者字符型。有两种变量函数。

调用指定属性标识符。

```
int  PlotIndexGetInteger(
   int  plot_index,         // 图样式指数
   int  prop_id,            // 属性标识符
   );
```

调用指定属性标识符和修饰符。

```
int  PlotIndexGetInteger(
   int  plot_index,         // 绘图指数
   int  prop_id,            // 属性标识符
   int  prop_modifier       // 属性修饰符
   )
```

参量

*plot_index*

[in] 图解 索引

*prop_id*

[in] 该值可以是 ENUM_PLOT_PROPERTY_INTEGER 列举值中的一个。

*prop_modifier*

[in] 规定属性的修饰符，只有显色指数性能要求修改。

注释

函数用来提取适当指示线路的绘画设置，该函数与PlotIndexSetInteger函数串联，通过复制绘图性质从一条线绘制到另一条线里。

示例：指标的色彩平台以周为单位。每天的颜色都在程序中体现。

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//---- 蜡烛颜色图
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标缓冲区
double          OpenBuffer[];
double          HighBuffer[];
double          LowBuffer[];
double          CloseBuffer[];
double          ColorCandlesColors[];
color           ColorOfDay[6]={CLR_NONE,clrMediumSlateBlue,
                               clrDarkGoldenrod,clrForestGreen,clrBlueViolet,clrRed};
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,OpenBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,HighBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LowBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,CloseBuffer,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- 颜色缓存区设置颜色数
```

© 2000-2025, MetaQuotes Ltd.

```
   PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,6);
//--- 为颜色缓存设置颜色
   for(int i=1;i<6;i++)
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,i,ColorOfDay[i]);
//--- 设置精确度
   IndicatorSetInteger(INDICATOR_DIGITS,_Digits);
   printf("We have %u colors of days",PlotIndexGetInteger(0,PLOT_COLOR_INDEXES));
//---
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   int i;
   MqlDateTime t;
//----
   if(prev_calculated==0) i=0;
   else i=prev_calculated-1;
//----
   while(i<rates_total)
     {
      OpenBuffer[i]=open[i];
      HighBuffer[i]=high[i];
      LowBuffer[i]=low[i];
      CloseBuffer[i]=close[i];
      //--- 为每个蜡烛设置颜色
      TimeToStruct(time[i],t);
      ColorCandlesColors[i]=t.day_of_week;
      //---
      i++;
     }
//--- 为下一次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# 物件函数

这是意于用有关特定图表的图解物件工作的函数组。

定义图解物件属性的函数，以及用于根据图表创建和移动物件的ObjectCreate()和ObjectMove()操作实际上是用来向图表发送命令的。 如果这些函数成功执行，那么这个命令就包含在图表事件的公共队列中。当处理图表事件的队列时，图解物件属性的可视变化将被实施。

因此，调用这些函数以后，不要期望立即对图解物件进行可视化更新。通常情况下，图表上的图解物件会在变更事件之后自动更新 – 进入新报价，调整图表窗口大小等。使用ChartRedraw()函数，强制更新图解物件。

| 函数 | 功能 |
|---|---|
| ObjectCreate | 创建指定图表中指定类型物件 |
| ObjectName | 返回指定图表中相关类型物件名称（指定图表子窗口） |
| ObjectDelete | 从指定图表移除指定名物件（从指定图表子窗口） |
| ObjectsDeleteAll | 从指定图表移除指定类型所有物件（从指定图表子窗口） |
| ObjectFind | 依据名称搜索指定ID物件 |
| ObjectGetTimeByValue | 对指定物件价格值返回时间值 |
| ObjectGetValueByTime | 对指定时间返回物件价格值 |
| ObjectMove | 改变指定物件定位点坐标 |
| ObjectsTotal | 返回指定图表中指定类型物件数（指定图表子窗口） |
| ObjectGetDouble | 返回相应物件属性的双精度值 |
| ObjectGetInteger | 返回相应物件属性的整数型值 |
| ObjectGetString | 返回相应物件属性的字符串值 |
| ObjectSetDouble | 设置相应物件属性值 |
| ObjectSetInteger | 设置相应物件属性值 |
| ObjectSetString | 设置相应物件属性值 |
| TextSetFont | 设置使用绘制方法展示文本的字体（默认使用Arial120） |
| TextOut | 文本转到专为创建图形资源而设计的自定义数组（缓冲区） |
| TextGetSize | 返回当前字体设置的字符串宽度和高度 |

每个图表物件都应该有唯一一个 图表名称， 包括子窗口。图表物件名称的改变生成2个事件：物件删除旧名称事件，创建物件新名称事件。

在新建物件或者修改 物件属性 之后，推荐调用 ChartRedraw() 函数，命令客户端强制绘图（可见物件仍涵盖在内）。

© 2000-2025, MetaQuotes Ltd.

# ObjectCreate

在制定图表子窗口中，函数创建指定名称，类型和原坐标物件，在创建期间可以指定到30个坐标。

```
bool  ObjectCreate(
   long          chart_id,       // 图表标识符
   string        name,           // 物件名称
   ENUM_OBJECT   type,           // 物件类型
   int           sub_window,     // 窗口索引
   datetime      time1,          // 第一定位点的时间
   double        price1,         // 第一定位点的价格
   ...
   datetime      timeN=0,        // 第N个定位点的时间
   double        priceN=0,       //
   ...
   datetime      time30=0,       // 第30个定位点的时间
   double        price30=0       // 第30个定位点的价格
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。名称在图表中是唯一的，包括子窗口。

*类型*

[in] 物件类型。值可以是 ENUM_OBJECT 值中的一个。

*sub_window*

[in] 图表子窗口数量。0代表主图表窗口，指定子窗口一定存在，否则函数返回错误值。

*time1*

[in] 第一定位点的时间坐标。

*price1*

[in] 第一定位点的价格坐标。

*timeN=0*

[in] N点时间坐标。

*priceN=0*

[in] N点的价格坐标。

*time30=0*

[in] 第三十定位点的时间坐标。

*price30=0*

[in] 第三十定位点的价格坐标。

© 2000-2025, MetaQuotes Ltd.

返回值

如果命令成功添加到指定图表队列那么函数返回true，否则返回false。如果对象已经创建，则会尝试改变它的坐标。

注释

非同步调用通常用于ObjectCreate()，这也是函数仅返回命令添加到图表队列的结果。在这种情况下，true仅表示命令已成功加入队列，但执行结果尚不可知。

若要检查命令执行结果，您可以使用ObjectFind()函数或任何其他请求对象属性的函数，例如ObjectGetXXX。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果（因为同步调用），因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

对象名称不应超过63个字符。

图表子窗口的数量（如果图表中有附带指标的子窗口）以1开始，图表主窗口总是以0开始检索。

大量定位点（到30）提供更多使用，于此同时限制图表中的30个可能的定位点，调用的大量参量不能超过64。

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

这里有在创建每个物件类型时必须指定的一定数量的定位点：

| ID | 描述 | 定位点 |
|---|---|---|
| OBJ_VLINE | 垂直线 | 一个定位点。实际上只使用时间坐标。 |
| OBJ_HLINE | 水平线 | 一个定位点。实际上只使用价格坐标。 |
| OBJ_TREND | 趋势线 | 两个定位点。 |
| OBJ_TRENDBYANGLE | 角度趋势线 | 两个定位点。 |
| OBJ_CYCLES | 循环线 | 两个定位点。 |
| OBJ_ARROWED_LINE | 箭头线 | 两个定位点。 |
| OBJ_CHANNEL | 等距通道 | 三个定位点。 |
| OBJ_STDDEVCHANNEL | 标准偏差通道 | 两个定位点。 |
| OBJ_REGRESSION | 线性回归通道 | 两个定位点。 |
| OBJ_PITCHFORK | 安德鲁鱼叉理论 | 三个定位点。 |
| OBJ_GANNLINE | 江恩线 | 两个定位点。 |
| OBJ_GANNFAN | 江恩扇形线 | 两个定位点。 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 定位点 |
| --- | --- | --- |
| OBJ_GANNGRID | 江恩网格 | 两个定位点。 |
| OBJ_FIBO | 斐波纳契回调线 | 两个定位点。 |
| OBJ_FIBOTIMES | 斐波纳契时间周期线 | 两个定位点。 |
| OBJ_FIBOFAN | 斐波纳契扇形线 | 两个定位点。 |
| OBJ_FIBOARC | 斐波纳契角度线 | 两个定位点。 |
| OBJ_FIBOCHANNEL | 斐波纳契通道 | 三个定位点。 |
| OBJ_EXPANSION | 斐波纳契扩展线 | 三个定位点。 |
| OBJ_ELLIOTWAVE5 | 埃利奥特波动 | 五个定位点。 |
| OBJ_ELLIOTWAVE3 | 埃利奥特修正波 | 三个定位点。 |
| OBJ_RECTANGLE | 矩形 | 两个定位点。 |
| OBJ_TRIANGLE | 三角形 | 三个定位点。 |
| OBJ_ELLIPSE | 椭圆形 | 三个定位点。 |
| OBJ_ARROW_THUMB_UP | 拇指向上 | 一个定位点。 |
| OBJ_ARROW_THUMB_DOWN | 拇指向下 | 一个定位点。 |
| OBJ_ARROW_UP | 箭头向上 | 一个定位点。 |
| OBJ_ARROW_DOWN | 箭头向下 | 一个定位点。 |
| OBJ_ARROW_STOP | 停止符号 | 一个定位点。 |
| OBJ_ARROW_CHECK | 检查符号 | 一个定位点。 |
| OBJ_ARROW_LEFT_PRICE | 向左价格标签 | 一个定位点。 |
| OBJ_ARROW_RIGHT_PRICE | 向右价格标签 | 一个定位点。 |
| OBJ_ARROW_BUY | 买入符号 | 一个定位点。 |
| OBJ_ARROW_SELL | 卖出符号 | 一个定位点。 |
| OBJ_ARROW | 箭头 | 一个定位点。 |
| OBJ_TEXT | 文本 | 一个定位点。 |
| OBJ_LABEL | 标签 | 使用 OBJPROP_XDISTANCE 和 OBJPROP_YDISTANCE 属性设置位置。 |
| OBJ_BUTTON | 按钮 | 使用 OBJPROP_XDISTANCE and OBJPROP_YDISTANCE属性设置位置。 |
| OBJ_CHART | 图表 | 使用 OBJPROP_XDISTANCE and OBJPROP_YDISTANCE属性设置位置。 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 | 定位点 |
| --- | --- | --- |
| OBJ_BITMAP | 位图 | 一个定位点。 |
| OBJ_BITMAP_LABEL | 位图标签 | 使用 OBJPROP_XDISTANCE and OBJPROP_YDISTANCE 属性设置位置。 |
| OBJ_EDIT | 编辑 | 使用OBJPROP_XDISTANCE and OBJPROP_YDISTANCE 属性设置位置。 |
| OBJ_EVENT | 经济日历中对应事件的"事件"物件 | 一个定位点。实际上只使用时间坐标。 |
| OBJ_RECTANGLE_LABEL | 用于创建和设计自定义图形界面的 "矩形标签"对象。 | 使用OBJPROP_XDISTANCE and OBJPROP_YDISTANCE 属性设置位置。 |

© 2000-2025, MetaQuotes Ltd.

# ObjectName

函数返回指定图表中相关物件名称，在制定子窗口中的指定类型。

```
string  ObjectName(
   long   chart_id,           // 图表标识符
   int    pos,                // 物件列表中的数量
   int    sub_window=-1,      // 窗口索引
   int    type=-1             // 物件类型
   );
```

参量

*chart_id*

[in] 图表标识符。0表示当前图表。

*pos*

[in] 根据指定过滤器物件的序列号确定子窗口的类型和号码。

*sub_window=-1*

[in] 图表子窗口的号码。0代表主图表窗口，-1代表所有图表子窗口，包括主窗口。

*type=-1*

[in] 物件类型。值可以是 ENUM_OBJECT 值中一个，-1表示全部类型。

返回值

如果成功返回物件名称。

注释

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

© 2000-2025, MetaQuotes Ltd.

# ObjectDelete

函数从指定图表中的指定名称里删除物件。

```
bool  ObjectDelete(
   long    chart_id,      // 图表标识符
   string  name           // 物件名称
   );
```

参量

*chart_id*

[in] 图表标识符。0表示当前图表。

*name*

[in] 删除物件的名称。

返回值

如果命令成功添加到指定图表队列那么函数返回true，否则返回false。

注释

非同步调用通常用于ObjectDelete()，这也是函数仅返回命令添加到图表队列的结果。在这种情况下，true仅表示命令已成功加入队列，但执行结果尚不可知。

若要检查命令执行结果，您可以使用ObjectFind()函数或任何其他请求对象属性的函数，例如ObjectGetXXX。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果（因为同步调用），因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

© 2000-2025, MetaQuotes Ltd.

# ObjectsDeleteAll

从指定图表中删除所有物件，指定图表子窗口，指定类型。

```
int  ObjectsDeleteAll(
   long  chart_id,           // 图表标识符
   int   sub_window=-1,      // 窗口索引
   int   type=-1             // 物件类型
   );
```

使用对象名称的前缀移除所有指定类型的对象。

```
int  ObjectsDeleteAll(
   long           chart_id,   // 图表标识符
   const string     prefix,   // 对象名称前缀
   int        sub_window=-1,  // 窗口索引
   int       object_type=-1   // 对象类型
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*prefix*

[in] 对象名称前缀：所有名称以这组字符开头的对象都将从图表移除。您可以指定前缀为'name'或'name*'-两种变量的工作相同。如果空字符串设定为前缀，那么所有可能名称的对象都将被移除。

*sub_window=-1*

[in] 图表子窗口号码。0代表主要图表窗口，-1代表图表子窗口，包括主窗口。

*type=-1*

[in] 物件类型。值可以是 ENUM_OBJECT 值中一个，-1表示全部类型。

返回值

返回删除物件的数量。阅读更多关于 错误 信息，调用 GetLastError()。

注意

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

# ObjectFind

在图表指定ID中函数搜索指定名称的物件。

```
int  ObjectFind(
   long    chart_id,      // 图表标识符
   string  name           // 物件名称
   );
```

参量

*chart_id*

[in] 图表标识符，0表示当前图表。

*name*

[in] 搜索物件名称。

返回值

如果成功函数返回子窗口的号码（0表示图表的主窗口），发现物件。如果没有找到物件，函数返回负值，阅读更多关于错误，调用 GetLastError()。

注释

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

© 2000-2025, MetaQuotes Ltd.

# ObjectGetTimeByValue

指定物件的指定价格值，函数返回时间值。

```
datetime  ObjectGetTimeByValue(
   long     chart_id,      // 图表标识符
   string   name,          // 物件名称
   double   value,         // 价格
   int      line_id        // 线
   );
```

参量

*chart_id*

[in] 图表标识符。0表示当前图表。

*name*

[in] 物件名称。

*value*

[in] 价格值。

*line_id*

[in] 线路标识符。

返回值

为指定物件的指定价格值设定时间值。

注释

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

一个对象在一个价格坐标内可以有多个值，因此有必要指明行号。这个函数仅应用于以下对象：

- 趋势线 (OBJ_TREND)
- 角度趋势线 (OBJ_TRENDBYANGLE)
- 江恩线 (OBJ_GANNLINE)
- 等距通道 (OBJ_CHANNEL) - 2行
- 线性回归通道 (OBJ_REGRESSION) - 3行
- 标准偏差通道 (OBJ_STDDEVCHANNEL) - 3行
- 箭头线 (OBJ_ARROW ED_LINE)

另见

对象类型

© 2000-2025, MetaQuotes Ltd.

# ObjectGetValueByTime

函数返回为指定物件指定时间值设定的价格值。

```
double  ObjectGetValueByTime(
   long      chart_id,     // 图表标识符
   string    name,         // 物件名称
   datetime  time,         // 时间
   int       line_id       // 线
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*time*

[in] 时间值。

*line_id*

[in] 线形ID。

返回值

指定物件的指定时间值的价格值。

注释

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

一个对象在一个价格坐标内可以有多个值，因此有必要指明行号。这个函数仅应用于以下对象：

- 趋势线(OBJ_TREND)
- 角度趋势线(OBJ_TRENDBYANGLE)
- 江恩线(OBJ_GANNLINE)
- 等距通道(OBJ_CHANNEL)-2行
- 线性回归通道 (OBJ_REGRESSION)-3行
- 标准偏差通道(OBJ_STDDEVCHANNEL)-3行
- 箭头线 (OBJ_ARROWED_LINE)

另见

对象类型

© 2000-2025, MetaQuotes Ltd.

# ObjectMove

函数物件定位点的指定坐标。

```
bool  ObjectMove(
   long      chart_id,         // 图表标识符
   string    name,             // 物件名称
   int       point_index,      // 定位点数
   datetime  time,             // 时间
   double    price             // 价格
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*point_index*

[in] 定位点检索。定位点数量取决于物件类型。

*time*

[in] 所选定位点的时间坐标。

*价格*

[in] 所选定位点的价格坐标。

返回值

如果命令成功添加到指定图表队列那么函数返回true，否则返回false。

注意

非同步调用通常用于ObjectMove()，这也是函数仅返回命令添加到图表队列的结果。在这种情况下，true仅表示命令已成功加入队列，但执行结果尚不可知。

若要检查命令执行结果，您可以使用请求对象属性的函数，例如ObjectGetXXX。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果（因为同步调用），因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

# ObjectsTotal

在指定类型的指定子窗口的指定图表物件中，函数返回物件数量。

```
int  ObjectsTotal(
   long  chart_id,            // 图表标识符
   int   sub_window=-1,       // 窗口索引
   int   type=-1              // 物件类型
   );
```

参量

*chart_id*

[in] 图表标识符。0表示当前图表。

*sub_window=-1*

[in] 图表子窗口的数量。0代表主图表窗口，-1代表图表的所有子窗口，包括主窗口。

*type=-1*

[in] 物件类型。值可以是 ENUM_OBJECT 值中一个，-1表示全部类型。

返回值

物件数量。

注意

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

# ObjectSetDouble

函数建立类似物件属性的值。物件数据应该是 双精度 类型，有2中类型函数可供使用。

设置属性值，无修饰符

```
bool  ObjectSetDouble(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // 属性
   double                        prop_value         // 值
   );
```

设置表明修饰符属性值

```
bool  ObjectSetDouble(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // 属性
   int                           prop_modifier,     // 修饰符
   double                        prop_value         // 值
   );
```

参量

*chart_id*

[in] 图表标识符。0表示当前图表。

*name*

[in] 物件名称。

*prop_id*

[in] 物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_DOUBLE 值中一个。

*prop_modifier*

[in] 指定属性修饰语，表示 斐波纳契工具 水平线数量和安德鲁分叉图解物件，水平线编号从0开始。

*prop_value*

[in] 属性值。

返回值

如果改变图解物件属性的命令成功发送到图表该函数返回true。否则返回false。调用函数 GetLastError() 读取更多关于错误信息。

注意

这个函数使用非同步调用，这意味着这个函数不等待执行已入列图表的命令。相反，而是立即返回控制。

© 2000-2025, MetaQuotes Ltd.

若要检查命令执行结果，您可以使用请求对象属性的函数。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果，因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

新建斐波纳契物件和添加新水平线示例

```
//+------------------------------------------------------------------+
//| 脚本程序启动函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 辅助数组
   double high[],low[],price1,price2;
   datetime time[],time1,time2;
//--- 复制开盘价 - 最新的100柱就足够
   int copied=CopyHigh(Symbol(),0,0,100,high);
   if(copied<=0)
     {
      Print("Failed to copy the values of the High price series");
      return;
     }
//--- 复制收盘价 - 最新100柱就足够
   copied=CopyLow(Symbol(),0,0,100,low);
   if(copied<=0)
     {
      Print("Failed to copy the values of the Low price series");
      return;
     }
//--- 复制最新100柱的开盘时间
   copied=CopyTime(Symbol(),0,0,100,time);
   if(copied<=0)
     {
      Print("Failed to copy the values of the price series of Time");
      return;
     }
//--- 如访问时间序列一样组织访问复制的数据-反向
   ArraySetAsSeries(high,true);
   ArraySetAsSeries(low,true);
   ArraySetAsSeries(time,true);

//--- 斐波纳契物件的第一定位点的坐标
   price1=high[70];
   time1=time[70];
//--- 斐波纳契物件的第二定位点的坐标
   price2=low[50];
   time2=time[50];

//--- 创建斐波纳契物件的时间
   bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2);
```

© 2000-2025, MetaQuotes Ltd.

```
   if(created) // 如果物件成功创建
     {
      //--- 设置斐波纳契色阶调整颜色
      ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
      //--- 另外，有多少斐波纳契色阶？
      int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
      Print("Fibo levels before = ",levels);
      //---输出到日志 => 水平号:水平值描述
      for(int i=0;i<levels;i++)
        {
         Print(i,": ",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
               "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
        }
      //--- 增加每单元的水平面数量
      bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
      if(!modified) // 更改水平面数量失败
        {
         Print("Failed to change the number of levels of Fibo, error ",GetLastError())
        }
      //--- 只是通知
      Print("Fibo levels after = ",ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS));
      //--- 为新创建的水平面设置一个值
      bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
      if(added) // 为水平面管理并设置一个值
        {
         Print("Successfully set one more Fibo level");
         //--- 还不要忘记设置水平面描述
         ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
         ChartRedraw(0);
         //--- 在斐波纳契物件中获得水平面数量的实际值
         levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
         Print("Fibo levels after adding = ",levels);
         //--- 再次输出所有水平-只是为了证实
         for(int i=0;i<levels;i++)
           {
            Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                  "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
           }
        }
      else // 如果在斐波纳契物件中增加水平面数量则会失败
        {
         Print("Failed to set one more Fibo level. Error ",GetLastError());
        }
     }
  }
```

另 见

物件类型, 物件属性

© 2000-2025, MetaQuotes Ltd.

# ObjectSetInteger

函数建立类似物件属性值，物件属性一定是日期时间，整型，颜色，布尔或者字符 类型。有2个变量函数可以使用。

设置属性值，无修饰符

```
bool  ObjectSetInteger(
   long                              chart_id,          // 图表标识符
   string                            name,              // 物件名称
   ENUM_OBJECT_PROPERTY_INTEGER      prop_id,           // 属性
   long                              prop_value         // 值
   );
```

设置表明修饰符属性值

```
bool  ObjectSetInteger(
   long                              chart_id,          // 图表标识符
   string                            name,              // 物件名称
   ENUM_OBJECT_PROPERTY_INTEGER      prop_id,           // 属性
   int                               prop_modifier,     // 修饰符
   long                              prop_value         // 值
   );
```

参量

*chart_id*

[in] 图表标识符，0代表当前图表。

*name*

[in] 物件名称。

*prop_id*

[in] 物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_INTEGER 值中的一个。

*prop_modifier*

[in] 指定属性修饰语。表示 斐波纳契工具 水平线数量和安德鲁分叉图解物件，水平线编号从0开始。

*prop_value*

[in] 属性值。

返回值

如果改变图解物件属性的命令成功发送到图表该函数返回true。否则返回false。调用函数 GetLastError() 读取更多关于错误信息。

注意

这个函数使用非同步调用，这意味着这个函数不等待执行已入列图表的命令。相反，而是立即返回控制。

© 2000-2025, MetaQuotes Ltd.

若要检查命令执行结果，您可以使用请求对象属性的函数。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果，因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

如何创建网络颜色表格的示例

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                Table of Web Colors|
//|                        Copyright 2011, MetaQuotes Software Corp |
//|                                             https://www.metaquotes.net |
//+------------------------------------------------------------------+
#define X_SIZE 140      // 编辑物件的宽度
#define Y_SIZE 33       // 编辑物件的高度
//+------------------------------------------------------------------+
//| 网络颜色数组                                                     |
//+------------------------------------------------------------------+
color ExtClr[140]=
  {
   clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,clrBisque,clrB
   clrBlue,clrBlueViolet,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,clrChocolate
   clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod,clrDarkGray
   clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed,clrDarkSalm
   clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDeepPink,clrDee
   clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clrGainsboro,cl
   clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clrIndianRed,cl
   clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,clrLightCora
   clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmon,clrLightSe
   clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen,clrLinen,cl
   clrMediumAquamarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMediumSeaGreen
   clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightBlue,clrMintC
   clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOrangeRed,clrO
   clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPuff,clrPeru,c
   clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clrSandyBrown,c
   clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpringGreen,cl
   clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteSmoke,clrYell
  };
//+------------------------------------------------------------------+
//| 创建和初始化编辑物件                                             |
//+------------------------------------------------------------------+
void CreateColorBox(int x,int y,color c)
  {
//--- 生成新编辑物件的名称
   string name="ColorBox_"+(string)x+"_"+(string)y;
//--- 创建新的编辑物件
   if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
     {
      Print("Cannot create: '",name,"'");
      return;
     }
//--- 设置坐标，宽度和高度
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x*X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y*Y_SIZE);
   ObjectSetInteger(0,name,OBJPROP_XSIZE,X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YSIZE,Y_SIZE);
//--- 设置文本颜色
   if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
   else            ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
//--- 设置背景色
   ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- 设置文本
   ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
  }
//+------------------------------------------------------------------+
//| 脚本程序开始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建颜色编辑物件的7x20表格
   for(uint i=0;i<140;i++)
      CreateColorBox(i%7,i/7,ExtClr[i]);
  }
```

另见

物件类型, 物件属性

© 2000-2025, MetaQuotes Ltd.

# ObjectSetString

函数建立类似于物件属性的值，物件属性必须是 字符串 类型。有2个变量函数可以使用。

设置属性值，无修饰符

```
bool  ObjectSetString(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_STRING   prop_id,           // 属性
   string                        prop_value         // 值
   );
```

设置表明修饰符属性值

```
bool  ObjectSetString(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_STRING   prop_id,           // 属性
   int                           prop_modifier,     // 修饰符
   string                        prop_value         // 值
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*prop_id*

[in] 物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_STRING 值中一个。

*prop_modifier*

[in] 指定属性修饰语，表示斐波纳契工具 水平线数量和安德鲁分叉图解物件，水平线编号从0开始。

*prop_value*

[in] 属性值。

返回值

如果改变图解物件属性的命令成功发送到图表该函数返回true。否则返回false。调用函数 GetLastError() 读取更多关于错误信息。

注释

这个函数使用非同步调用，这意味着这个函数不等待执行已入列图表的命令。相反，而是立即返回控制。

© 2000-2025, MetaQuotes Ltd.

若要检查命令执行结果，您可以使用请求对象属性的函数。但是，您需要牢记的是这类函数将被加入到图表的队尾，需要等待执行结果，因此，可能会耗费大量时间。当处理图表上的大量对象时应该考虑这个特性。

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

© 2000-2025, MetaQuotes Ltd.

# ObjectGetDouble

函数返回类似物件属性的返回值，物件属性必须是双精度类型。有2个变量函数可以使用。

1. 即时返回属性值。

```
double  ObjectGetDouble(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // 属性标识符
   int                           prop_modifier=0    // 属性修饰符，如果需要的话
   );
```

2. 返回 true 或者 false，取决于函数是否成功。如果成功，属性值通过上一参量以引用的方式传递安置接收变量。

```
bool  ObjectGetDouble(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // 属性标识符
   int                           prop_modifier,     // 属性修饰符
   double&                       double_var         // 这里接受属性值
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*prop_id*

[in] 物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_DOUBLE 值中一个。

*prop_modifier*

[in] 指定属性修饰语，第一变体，默认修饰语的值是0,大多属性不要求修饰语。表示斐波纳契工具水平线数量和安德鲁分叉图解物件，水平线编号从0开始。

*double_var*

[out] 双精度类型变量接收需求属性值。

返回值

第一次调用变体的双精度类型返回值。

第二变体的函数返回真值，如果该属性是被维护的并且值可以放置到double_var 中，否则返回错误值。为了阅读更多关于错误，调用GetLastError()。

注意

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

# ObjectGetInteger

函数返回类似物件属性值，物件属性必须是日期时间，整型，颜色，布尔或者字符 类型。有2个变量函数可以使用。

1.即时返回属性值。

```
long  ObjectGetInteger(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,           // 属性标识符
   int                           prop_modifier=0    // 属性修饰符, 如果需要的话
   );
```

2. 返回 true 或者 false, 取决于函数是否成功。如果成功，属性值通过上一参量以引用的方式传递安置接收变量。

```
bool  ObjectGetInteger(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   int                           prop_id,           // 属性标识符
   ENUM_OBJECT_PROPERTY_INTEGER  prop_modifier,     // 属性修饰符
   long&                         long_var           // 这里接受属性值
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*prop_id*

[in] 物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_INTEGER 值中的一个。

*prop_modifier*

[in] 指定属性修饰语，第一变体，默认修饰语的值是0,。大多属性不要求修饰语。表示斐波纳契工具 水平线数量和安德鲁分叉图解物件，水平线编号从0开始。

*long_var*

[out] 长型变量接收要求属性值。

返回值

第一次调用变体的长类型返回值。

第二变体的函数返回真值，如果该属性是被维护的并且值可以放置到long_var 中，否则返回错误值。为了阅读更多关于 错误，调用 GetLastError()。

注意

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

# ObjectGetString

函数返回类似物件属性值，物件属性必须是 字符串 类型，有2个变量函数可以使用。

1.即时返回属性值。

```
string  ObjectGetString(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_STRING   prop_id,           // 属性标识符
   int                           prop_modifier=0    // 属性修饰符, 如果需要的话
   );
```

2. 返回 true 或者 false, 取决于函数是否成功。 如果成功，属性值通过上一参量以引用的方式传递安置接收变量。

```
bool  ObjectGetString(
   long                          chart_id,          // 图表标识符
   string                        name,              // 物件名称
   ENUM_OBJECT_PROPERTY_STRING   prop_id,           // 属性标识符
   int                           prop_modifier,     // 属性修饰符
   string&                       string_var         // 这里接受属性值
   );
```

参量

*chart_id*

[in] 图表标识符。0代表当前图表。

*name*

[in] 物件名称。

*prop_id*

[in]物件属性ID。值可以是 ENUM_OBJECT_PROPERTY_STRING 值中的一个。

*prop_modifier*

[in] 指定属性修饰语，第一变体，默认修饰语的值是0,大多属性不要求修饰语。表示斐波纳契工具 水平线数量和Andrew's pitchfork图解物件，水平线编号从0开始。

*string_var*

[out] 字符串类型变量接收要求属性值。

返回值

第一次调用版本的字符串值。

第二变体的函数返回真值，如果该属性是被维护的并且值可以放置到string_var 中，否则返回错误值。为了阅读更多关于 错误 ，调用 GetLastError()。

注释

该函数使用同步调用，这意味着这个函数等待执行在调用之前已入列图表的所有命令，这就是该函数耗费时间的原因。当处理图表上的大量对象时应该考虑这个特性。

© 2000-2025, MetaQuotes Ltd.

当物件重命名，两个事件同时形成。这些事件可以在EA交易或者 OnChartEvent() 函数指标中处理：

- 旧名称物件删除事件；
- 新名称物件新建事件。

© 2000-2025, MetaQuotes Ltd.

# TextSetFont

该函数使用绘制方法设置展示文本的字体并返回操作结果。默认使用-120 (12 pt)大小的Aria字体。

```
bool  TextSetFont(
   const string  name,              // 字体名称或磁盘上字体文件的路径
   int           size,              // 字体大小
   uint          flags,             // 混合标识
   int           orientation=0      // 文本方向
   );
```

参数

*name*

[in] 系统字体名称或包括字体或磁盘字体文件路径的资源名称。

*size*

[in] 字体大小。

*flags*

[in] 描述字体样式的混合标识 。

*orientation*

[in] X轴的文本水平倾斜，测量单位0.1度。它表示*方向=450*象征倾斜45度。

返回值

如果当前字体成功安装，返回true，否则返回false。可能的错误代码：

- ERR_INVALID_PARAMETER(4003) - *name* 显示 NULL 或 "" (空字符串)，
- ERR_INTERNAL_ERROR(4001) - 操作系统错误 (例如，试图创建不存在的字体)。

注意

如果"::" 被用作字体名称，字体从 EX5 资源下载。如果*name*字体名称扩展指定，字体从文件下载，如果路径从"\"或"/"开始，文件相对MQL5目录搜索。否则，相对调用TextSetFont()函数的EX5文件路径搜索。

字体大小使用正负值进行设置。该事实从操作系统设置定义了文本大小的依赖性 (大小比例)。

- 如果大小指定正数，那么当从逻辑字体变为物理字体时，该大小转变为设备的物理测量单位 (像素)，并且该大小符合从有效字体挑选的符号图形高度。当在图表上一起使用通过TextOut(函数展示的文本和通过OBJ_LABEL ("标签") 图形物件展示的文本时，这种情况不被推荐。
- 如果大小指定负数，那么该数字被认为是逻辑点的十分之一 (-350等于35逻辑点)并且被10除。获得的值然后转变为设备的物理测量单位 (像素)，并且符合从有效字体挑选的符号高度的绝对值。物件属性中指定的字体大小乘以-10，使得屏幕上的文本大小类似OBJ_LABEL物件的文本大小。

该标识可以被用作样式标识和指定字体宽度标识的结合体。标识名称显示如下。

指定字体样式的标识

© 2000-2025, MetaQuotes Ltd.

| 标识 | 描述 |
| --- | --- |
| FONT_ITALIC | 斜体字 |
| FONT_UNDERLINE | 下划线 |
| FONT_STRIKEOUT | 加删除线 |

指定字体宽度的标识

| 标识 |
| --- |
| FW_DONTCARE |
| FW_THIN |
| FW_EXTRALIGHT |
| FW_ULTRALIGHT |
| FW_LIGHT |
| FW_NORMAL |
| FW_REGULAR |
| FW_MEDIUM |
| FW_SEMIBOLD |
| FW_DEMIBOLD |
| FW_BOLD |
| FW_EXTRABOLD |
| FW_ULTRABOLD |
| FW_HEAVY |
| FW_BLACK |

另见

资源, ResourceCreate(), ResourceSave(), TextOut()

© 2000-2025, MetaQuotes Ltd.

# TextOut

该函数显示自定义数组中的文本（缓冲区）并返回该操作结果。数组是专为创建图形资源而设计的。

```
bool  TextOut(
   const string        text,          // 显示的文本
   int                 x,             // X 坐标
   int                 y,             // Y 坐标
   uint                anchor,        // 定位类型
   uint                &data[],       // 输出缓冲区
   uint                width,         // 缓冲区像素宽度
   uint                height,        // 缓冲区像素高度
   uint                color,         // 文本颜色
   ENUM_COLOR_FORMAT   color_format   // 输出的颜色格式
   );
```

参数

*text*

[in] 将被写入缓冲区的文本显示。只显示单行文本。

*x*

[in] 显示文本的定位点的 X 坐标。

*y*

[in] 显示文本的定位点的 Y 坐标。

*anchor*

[in] 显示文本定位点位置的9种预定义方法的值。该值通过结合两种标识设置 - 水平和垂直文本对齐标识。标识名称列在附注。

*data[]*

[in] 缓冲区，文本在这里显示。缓冲区被用于创建图形资源。

*width*

[in] 缓冲区像素宽度。

*height*

[in] 缓冲区像素高度。.

*color*

[in] 文本颜色。

*color_format*

[in] 通过ENUM_COLOR_FORMAT 枚举值设置的颜色格式。

返回值

如果成功，返回 true，否则 false。

注意

© 2000-2025, MetaQuotes Ltd.

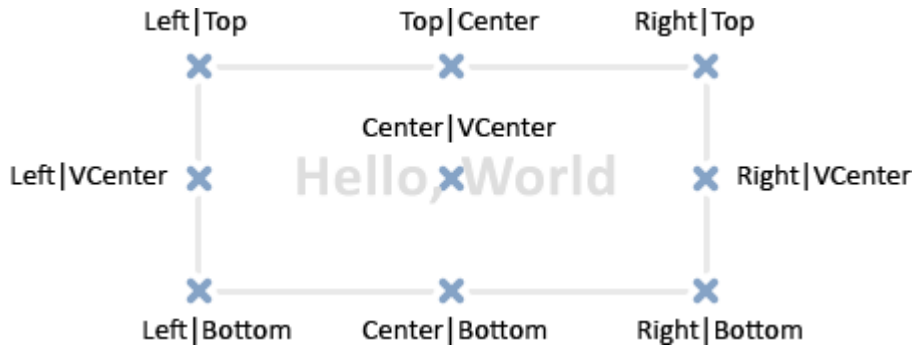
通过*anchor*指定的定位点是水平文本对齐和垂直文本对齐两个标识的组合。水平文本对齐标识：

- TA_LEFT -边框左侧定位点
- TA_CENTER -边框中心的水平定位点
- TA_RIGHT -边框右侧定位点

垂直文本对齐标识

- TA_TOP -边框上方定位点
- TA_VCENTER -边框中心的垂直定位点
- TA_BOTTOM -边框下方定位点

图片中显示了标识和指定定位点可能的组合。




例如：

```
//--- 画布的宽和高 用于绘制)
#define IMG_WIDTH  200
#define IMG_HEIGHT 200
//--- 启动脚本前显示参数窗口
#property script_show_inputs
//--- 启用设置颜色格式
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- 绘制数组 缓冲区)
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建用于绘制的OBJ_BITMAP_LABEL物件
   ObjectCreate(0,"CLOCK",OBJ_BITMAP_LABEL,0,0,0);
//--- 指定用于在CLOCK物件编写的图形资源的名称
   ObjectSetString(0,"CLOCK",OBJPROP_BMPFILE,"::IMG");

//--- 辅助变量
   double a;             // 箭头计数器
   uint   nm=2700;       // 分钟计数器
   uint   nh=2700*12;    // 小时计数器
   uint   w,h;           // 接收文本字符串大小的变量
```

© 2000-2025, MetaQuotes Ltd.

```
   int     x,y;           // 计算文本字符串定位点当前坐标的变量

//--- 无限循环旋转时针，直至脚本停止
   while(!IsStopped())
     {
      //--- 清空绘制缓冲区数组的时钟
      ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
      //--- 为钟面设置绘制数字的字体
      TextSetFont("Arial",-200,FW_EXTRABOLD,0);
      //--- 绘制钟面
      for(int i=1;i<=12;i++)
        {
         //--- 接收钟面上当前小时的大小
         TextGetSize(string(i),w,h);
         //--- 计算钟面上当前小时的坐标
         a=-((i*300)%3600*M_PI)/1800.0;
         x=IMG_WIDTH/2-int(sin(a)*80+0.5+w/2);
         y=IMG_HEIGHT/2-int(cos(a)*80+0.5+h/2);
         //--- 输出钟面上的小时到ExtImg[]缓冲区
         TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,
        }
      //--- 现在，指定绘制分针的字体
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nm%3600));
      //--- 接收分针的大小
      TextGetSize("----->",w,h);
      //--- 计算钟面上分针的坐标
      a=-(nm%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- 在ExtImg[]缓冲区输出分针到钟面
      TextOut("----->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_

      //--- 现在，设置绘制分针的字体
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nh/12%3600));
      TextGetSize("==>",w,h);
      //--- 计算钟面上时针的坐标
      a=-(nh/12%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- 在ExtImg[]缓冲区输出时针到钟面
      TextOut("==>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_forn

      //--- 更新图形资源
      ResourceCreate("::IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr_format);
      //--- 强制图表更新
      ChartRedraw();

      //--- 增加小时和分钟计数器
      nm+=60;
      nh+=60;
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 框架之间保持短暂停顿
      Sleep(10);
     }
//--- 完成脚本操作时删除 CLOCK 物件
   ObjectDelete(0,"CLOCK");
//---
  }
```

另 见

资源, ResourceCreate(), ResourceSave(), TextGetSize(), TextSetFont()

© 2000-2025, MetaQuotes Ltd.

# TextGetSize

该函数在当前字体设置返回线型宽度和高度。

```
bool  TextGetSize(
   const string       text,          // 文本字符串
   uint&               width,        // 缓冲区像素宽度
   uint&               height        // 缓冲区像素高度
   );
```

参数

*text*

[in] 字符串，应该用于获得长度和宽度。

*width*

[out] 接收宽度的输入参数。

*height*

[out] 接收高度的输入参数。

返回值

如果成功返回true，否则 false。可能的错误代码：

- ERR_INTERNAL_ERROR(4001) - 操作系统错误。

另见

资源, ResourceCreate(), ResourceSave(), TextSetFont(), TextOut()

© 2000-2025, MetaQuotes Ltd.

# 技术指标函数

所有函数，像iMA，iAC，iMACD，iIchimoku等，在客户端全局缓存区建立类似技术指标的副本，如果类似参量的指标副本已经存在，新的副本就不会建立，只是参考存在的副本增添项目。

这些函数返回适当的指标副本处理，使用该处理系统，可以通过类似指标收到计算数据。类似缓冲区数据（技术指标包括内部缓冲区的计算数据1到5的不同取决于指标）可以使用CopyBuffer()函数被MQL5程序复制。

不能参考新建的指标数据，因为指标计算值需要时间，因此最好在OnInit()中建立指标处理器，函数iCustom()创建类似自定义指标，返回的处理器就能成功建立。自定义指标包含最高512指标缓冲区，CopyBuffer()函数中包括目录，使用获得的处理器。

使用IndicatorCreate()函数创建任意计算指标是一种通用的办法，该函数以输入参量形式接收如下数据：

- 交易品种名称；
- 时间表；
- 创建的指标类型；
- 指标输入参量数量；
- MqlParam类型数组包含所有必要的输入参量。

计算机内存可以通过删除不使用的指标释放，使用IndicatorRelease()函数，传递指标处理器。

**注释。** 在MQL5程序中反复以相同参量调用指标函数不能导致参考计算器的增加；计算器值能被1建立一次，推荐使用OnInit()函数获得指标处理器，在其余函数中使用这些处理器，当MQL5程序初始化时，指标开始减少。

所有指标函数都有至少2个参量-交易品种和周期。交易品种NULL值代表当前交易品种，周期0值代表当前时间表。

| 函数 | 返回指标处理器： |
|---|---|
| iAC | 加速震荡指标 |
| iAD | 累积/分配 |
| iADX | 平均定向指数 |
| iADXWilder | 韦尔达平均定向指数 |
| iAlligator | 鳄鱼指标 |
| iAMA | 适合的移动平均数 |
| iAO | 动量震荡指标 |
| iATR | 平均真实区域 |
| iBearsPower | 熊市 |
| iBands | 布林带 |
| iBullsPower | 牛市 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 返回指标处理器： |
| --- | --- |
| iCCI | 商品通道指数 |
| iChaikin | 蔡金摆动指标 |
| iCustom | 自定义指标 |
| iDEMA | 双指数移动平均线 |
| iDeMarker | 指标 |
| iEnvelopes | 轨道线指标 |
| iForce | 强力指数 |
| iFractals | 分形学 |
| iFrAMA | 分形学适合移动平均数 |
| iGator | 鳄鱼振荡器 |
| iIchimoku | 一目均衡图 |
| iBWMFI | 威廉姆斯的市场便利指标 |
| iMomentum | 动量指标 |
| iMFI | 货币流量指标 |
| iMA | 移动平均数 |
| iOsMA | 移动平均振荡指标（MACD柱状图） |
| iMACD | 移动平均聚散指标 |
| iOBV | 平衡交易量 |
| iSAR | 抛物转向系统 |
| iRSI | 相对强弱指标 |
| iRVI | 相对活力指标 |
| iStdDev | 标准偏差 |
| iStochastic | 随机摆动指标 |
| iTEMA | 三倍指数移动平均数 |
| iTriX | 三倍指数移动平均数振荡指标 |
| iWPR | 威廉指数 |
| iVIDyA | 动态平均数便利指标 |
| iVolumes | 成交量 |

© 2000-2025, MetaQuotes Ltd.

# iAC

函数在客户端的全局缓存器建立 加速振荡指标并返回处理器，只有一个缓冲区。

```
int  iAC(
   string           symbol,     // 交易品种名称
   ENUM_TIMEFRAMES  period      // 周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，该数据应该用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中一个，0代表当前时间表。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAC.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAC technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- iAC标图
#property indicator_label1  "iAC"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   Call_iAC,                // 使用iAC
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iAC;          // 函数类型
input string                symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double iACBuffer[];
double iACColors[];
//--- 存储iAC 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在加速震荡指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iACBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iACColors,INDICATOR_COLOR_INDEX);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iAC)
      handle=iAC(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AC);
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iAC indicator for the symbol %s/%s,
                  name,
```

© 2000-2025, MetaQuotes Ltd.

```
                  EnumToString(period),
                  GetLastError());
       //--- 指标提前停止
       return(INIT_FAILED);
      }
//--- 显示加速震荡指标计算的交易品种/时间帧
   short_name=StringFormat("iAC(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iAC 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iAC指标数量值更改
//---或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iACBuffer 数组大于交易品种/周期iAC 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 用来自加速震荡指标的值填充iACBuffer 和 iACColors 数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffer(iACBuffer,iACColors,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住动量震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iAC 指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],          // 加速震荡指标值指标缓冲区
                          double &color_indexes[], // 颜色缓冲区 用于存储颜色标引）
                          int ind_handle,           // iAC 指标的处理程序
                          int amount                // 复制值的数量
                          )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分 iACBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAC indicator, error code %d",GetLastE
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 现在复制颜色标引
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy color values from the iAC indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iAD

函数返回累积/分配指标处理器。只有一个缓冲区。

```
int  iAD(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   ENUM_APPLIED_VOLUME  applied_volume       // 用于计算的交易量类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值表示当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中一个，0代表当前时间表。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 值中的一个。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图 iAD
#property indicator_label1  "iAD"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAD,               // 使用iAD
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iAD;          // 函数类型
input ENUM_APPLIED_VOLUME   volumes;                // 使用的交易量
input string                symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double          iADBuffer[];
//--- 存储 iAD 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在离散指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iADBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iAD)
      handle=iAD(name,period,volumes);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].integer_value=volumes;
      handle=IndicatorCreate(name,period,IND_AD,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iAD indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示离散指标计算的交易品种/时间帧
   short_name=StringFormat("iAD(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iAD 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iAD指标数量值更改
//---或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iADBuffer 数组大于交易品种/周期iAD 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
```

© 2000-2025, MetaQuotes Ltd.

```
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 用来自离散指标的值填充iADBuffer数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iADBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住离散指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iAD 指标的指标缓冲区                                          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // 离散指标线的指标缓冲区
                         int ind_handle,     // iAD指标的处理程序
                         int amount          // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iADBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAD indicator, error code %d",GetLastE
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                  |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iADX

函数返回平均定向移动指数指标处理器。

```
int  iADX(
   string           symbol,          // 交易品种名称
   ENUM_TIMEFRAMES  period,          // 周期
   int              adx_period       // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*adx_period*

[in] 周期计算指数。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0 - MAIN_LINE，1 - PLUSDI_LINE，2 - MINUSDI_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                    Demo_iADX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
//--- 标图 ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 标图 DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 标图 DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADX,              // 使用iADX
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iADX;          // 函数类型
input int                   adx_period=14;           // 计算周期
input string                symbol=" ";              // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // 时间帧
//--- 指标缓冲区
double          ADXBuffer[];
double          DI_plusBuffer[];
double          DI_minusBuffer[];
//--- 存储iADX 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在平均方向移动指数指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iADX)
      handle=iADX(name,period,adx_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADX,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iADX indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示平均方向移动指数指标计算的交易品种/时间帧
   short_name=StringFormat("iADX(%s/%s period=%d)",name,EnumToString(period),adx_perio
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iADX 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iADX 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iADXBuffer 数组大于交易品种/周期 iADX 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以平均方向移动指数指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_c
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住平均方向移动指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iADX指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // ADX 线的指标缓冲区
                           double &DIplus_values[],   // DI+指标缓冲区
                           double &DIminus_values[],  // DI- 指标缓冲区
```

© 2000-2025, MetaQuotes Ltd.

```
                              int ind_handle,            // iADXWilder 指标的处理程序
                              int amount                 // 复制值的数量
                              )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分 iADXBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分 DI_plusBuffer 数组
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分 DI_plusBuffer 数组
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iADXWilder

函数返回韦尔达平均定向移动指数处理器。

```
int  iADXWilder(
   string           symbol,          // 交易品种名称
   ENUM_TIMEFRAMES  period,          // 周期
   int              adx_period       // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*adx_period*

[in] 周期计算指数。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0-MAIN_LINE，1-PLUSDI_LINE，2-MINUSDI_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                  iADXWilder.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADXWilder technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
//--- 标图 ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
```

```
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 标图 DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 标图 DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADXWilder,         // 使用iADXWilder
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iADXWilder;   // 函数类型
input int                   adx_period=14;          // 计算周期
input string                symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double          ADXBuffer[];
double          DI_plusBuffer[];
double          DI_minusBuffer[];
//--- 存储iADXWilder 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在平均方向移动指数Welles Wilder指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iADXWilder)
      handle=iADXWilder(name,period,adx_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADXW,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iADXWilder indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示平均方向移动指数Welles Wilder指标计算的交易品种/时间帧
   short_name=StringFormat("iADXWilder(%s/%s period=%d)",name,EnumToString(period),adx
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iADXWilder 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iADXWilder 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iADXBuffer 数组大于交易品种/周期 iADXWilder 指标的数量值，那么我们不会复制任何
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以平均方向移动指数Welles Wilder指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_c
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住平均方向移动指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iADXWilder 指标的指标缓冲区                                  |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // ADX 线的指标缓冲区
                           double &DIplus_values[],   // DI+指标缓冲区
                           double &DIminus_values[],  // DI- 指标缓冲区
```

© 2000-2025, MetaQuotes Ltd.

```
                              int ind_handle,              // iADXWilder 指标的处理程序
                              int amount                   // 复制值的数量
                              )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分 iADXBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分 DI_plusBuffer 数组
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分 DI_plusBuffer 数组
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iAlligator

函数返回鳄鱼指标处理器。

```
int  iAlligator(
   string               symbol,           // 交易品种名称
   ENUM_TIMEFRAMES      period,           // 周期
   int                  jaw_period,       // 咽喉计算周期
   int                  jaw_shift,        // 咽喉平移
   int                  teeth_period,     // 牙齿计算周期
   int                  teeth_shift,      // 牙齿平移
   int                  lips_period,      // 唇部计算周期
   int                  lips_shift,       // 唇部平移
   ENUM_MA_METHOD       ma_method,        // 平滑类型
   ENUM_APPLIED_PRICE   applied_price     // 价格或者处理器类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*jaw_period*

[in] 蓝线的平均周期(鳄鱼颌骨)

*jaw_shift*

[in]关于价格图表的蓝线转换。

*teeth_period*

[in] 红线的平均周期(鳄鱼牙)。

*teeth_shift*

[in]关于价格图表的红线转换。

*lips_period*

[in] 绿线的平均周期 (鳄鱼唇部)。

*lips_shift*

[in]关于价格图表的绿线转换。

*ma_method*

[in] 求平均值的方式，可以是 ENUM_MA_METHOD 值中任意值。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

© 2000-2025, MetaQuotes Ltd.

注释

缓冲区代码如下：0 - GATORJAW _LINE，1 - GATORTEETH_LINE，2 - GATORLIPS_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iAlligator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAlligator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Alligator

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- 标图咽喉
#property indicator_label1  "Jaws"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 标图牙齿
#property indicator_label2  "Teeth"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 标图唇部
#property indicator_label3  "Lips"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrLime
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAlligator,         // 使用iAlligator
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
```

```
//--- 输入参数
input Creation              type=Call_iAlligator;    // 函数类型
input string                symbol=" ";              // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // 时间帧
input int                   jaw_period=13;           // 咽喉线周期
input int                   jaw_shift=8;             // 咽喉线移动
input int                   teeth_period=8;          // 牙齿线周期
input int                   teeth_shift=5;           // 牙齿线移动
input int                   lips_period=5;           // 唇部线周期
input int                   lips_shift=3;            // 唇部线移动
input ENUM_MA_METHOD        MA_method=MODE_SMMA;     // 鳄鱼平均线的方法
input ENUM_APPLIED_PRICE    applied_price=PRICE_MEDIAN;// 用于计算鳄鱼的价格类型
//--- 指标缓冲区
double          JawsBuffer[];
double          TeethBuffer[];
double          LipsBuffer[];
//--- 存储 iAlligator 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在鳄鱼指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,JawsBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,TeethBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LipsBuffer,INDICATOR_DATA);
//--- 设置每条线的移动
   PlotIndexSetInteger(0,PLOT_SHIFT,jaw_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,teeth_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,lips_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
```

© 2000-2025, MetaQuotes Ltd.

```
   if(type==Call_iAlligator)
      handle=iAlligator(name,period,jaw_period,jaw_shift,teeth_period,
                        teeth_shift,lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[8];
     //--- 鳄鱼线的周期和移动
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
//--- 平滑类型
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
//--- 价格类型
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
//--- 创建处理程序
      handle=IndicatorCreate(name,period,IND_ALLIGATOR,8,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iAlligator indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示鳄鱼指标计算的交易品种/时间帧
   short_name=StringFormat("iAlligator(%s/%s, %d,%d,%d,%d,%d,%d)",name,EnumToString(pe
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,l
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iAlligator 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iAlligator指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果JawsBuffer 数组大于交易品种/周期 iAlligator 指标的数量值，那么我们不会复制任何
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以鳄鱼指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(JawsBuffer,jaw_shift,TeethBuffer,teeth_shift,LipsBuffer,l
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住鳄鱼指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
```

© 2000-2025, MetaQuotes Ltd.

```
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iAlligator指标的指标缓冲区                                  |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &jaws_buffer[],  // 咽喉线的指标缓冲区
                           int j_shift,            // 咽喉线移动
                           double &teeth_buffer[], // 牙齿线的指标缓冲区
                           int t_shift,            // 牙齿线移动
                           double &lips_buffer[],  // 唇部线的指标缓冲区
                           int l_shift,            // 唇部线移动
                           int ind_handle,         // iAlligator 指标的处理程序
                           int amount              // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分JawsBuffer 数组
   if(CopyBuffer(ind_handle,0,-j_shift,amount,jaws_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分TeethBuffer 数组
   if(CopyBuffer(ind_handle,1,-t_shift,amount,teeth_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分 LipsBuffer 数组
   if(CopyBuffer(ind_handle,2,-l_shift,amount,lips_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",G
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iAMA

函数返回适当移动平均指标处理器。只有一个缓冲区。

```
int  iAMA(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  ama_period,          //  AMA平均周期
   int                  fast_ma_period,      // 快速 MA 周期
   int                  slow_ma_period,      // 慢速 MA 周期
   int                  ama_shift,           // 指标平移
   ENUM_APPLIED_PRICE   applied_price        // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ama_period*

[in] 计算周期，计算效率系数。

*fast_ma_period*

[in] 在快速市场为平滑系数计算快速周期。

*slow_ma_period*

[in] 在缺乏趋势时为平滑系数计算的慢周期。

*ama_shift*

[in] 转换关于价格图表的指标。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                    Demo_iAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "of indicator buffers for the iAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard AMA."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图 iAMA
#property indicator_label1  "iAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAMA,              // 使用iAMA
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iAMA;          // 函数类型
input string                symbol=" ";              // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // 时间帧
input int                   ama_period=15;           // 计算周期
input int                   fast_ma_period=2;        // 快速MA周期
input int                   slow_ma_period=30;       // 慢速MA周期
input int                   ama_shift=0;             // 水平移动
input ENUM_APPLIED_PRICE    applied_price;           // 价格类型
//--- 指标缓冲区
double          iAMABuffer[];
//--- 存储 iAMA 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在适当移动平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区映射
   SetIndexBuffer(0,iAMABuffer,INDICATOR_DATA);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ama_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iAMA)
      handle=iAMA(name,period,ama_period,fast_ma_period,slow_ma_period,ama_shift,appli
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[5];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ama_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=fast_ma_period;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=slow_ma_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ama_shift;
      //--- 价格类型
      pars[4].type=TYPE_INT;
      pars[4].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_AMA,5,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iAMA indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示适当移动平均指标计算的交易品种/时间帧
   short_name=StringFormat("iAMA(%s/%s,%d,%d,%d,d)",name,EnumToString(period),ama_peri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iAMA 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iAMA 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iAMABuffer 数组大于交易品种/周期iAMA指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以适应移动平均指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iAMABuffer,ama_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 记住适当移动平均指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iAMA指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  //AMA线的指标缓冲区
                         int a_shift,           // AMA线移动
                         int ind_handle,        // iAMA 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分 iAMABuffer 数组
   if(CopyBuffer(ind_handle,0,-a_shift,amount,ama_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAMA indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iAO

函数返回动量震荡指标处理器。只有一个缓冲区。

```
int  iAO(
   string           symbol,     // 交易品种名称
   ENUM_TIMEFRAMES  period      // 周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAO.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAO technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- iAO 标图
#property indicator_label1  "iAO"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen,clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   Call_iAO,                  // 使用iAO
   Call_IndicatorCreate       // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iAO;           // 函数类型
input string                symbol=" ";              // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // 时间帧
//--- 指标缓冲区
double          iAOBuffer[];
double          iAOColors[];
//--- 存储 iAO指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在动量震荡指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iAOBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iAOColors,INDICATOR_COLOR_INDEX);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iAO)
      handle=iAO(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AO);
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iAO indicator for the symbol %s/%s,
                  name,
```

© 2000-2025, MetaQuotes Ltd.

```
                  EnumToString(period),
                  GetLastError());
       //--- 指标提前停止
       return(INIT_FAILED);
      }
//--- 显示动量震荡指标计算的交易品种/时间帧
   short_name=StringFormat("iAO(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iAO 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iAO 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iAOBuffer数组大于交易品种/周期iAO指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 用来自动量震荡指标的值填充iAOBuffer 和 iAOColors数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffer(iAOBuffer,iAOColors,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住动量震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iAO 指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],          //动量震荡指标值的指标缓冲区
                          double &color_indexes[], // 颜色缓冲区 用于存储颜色标引）
                          int ind_handle,          // iAO 指标的处理程序
                          int amount               // 复制值的数量
                          )
  {
//--- 重置错误代码
   ResetLastError();
//---以0标引指标缓冲区的值填充部分iAOBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iAO indicator, error code %d",GetLastE
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 现在复制颜色标引
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy color values from the iAO indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iATR

函数返回平均真实区域指标处理器。只有一个缓冲区。

```
int  iATR(
   string           symbol,        // 交易品种名称
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算平均周期值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                    Demo_iATR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iATR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图 iATR
#property indicator_label1  "iATR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iATR,// 使用 iATR
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input int                  atr_period=14;          // 计算周期
input Creation             type=Call_iATR;         // 函数类型
input string               symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double         iATRBuffer[];
//--- 存储iAC 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在平均真实区域指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iATRBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iATR)
      handle=iATR(name,period,atr_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].integer_value=atr_period;
      handle=IndicatorCreate(name,period,IND_ATR,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iATR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示平均真实范围指标计算的交易品种/时间帧
   short_name=StringFormat("iATR(%s/%s, period=%d)",name,EnumToString(period),atr_peri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iATR指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iATR 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iATRBuffer 数组大于交易品种/周期iATR指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
```

© 2000-2025, MetaQuotes Ltd.

```
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以平均真实波动范围指标的值填充 iATRBuffer 数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iATRBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住动量震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iATR 指标的指标缓冲区                                          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // ATR值的指标缓冲区
                         int ind_handle,    // iATR指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iATRBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iATR indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                  |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iBearsPower

函数返回熊市指标处理器。只有一个缓冲区。

```
int  iBearsPower(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算平均周期值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iBearsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBearsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iBearsPower 标图
#property indicator_label1  "iBearsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBearsPower,       // 使用iBearsPower
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iBearsPower;  // 函数类型
input int                  ma_period=13;           // 平均移动周期
input string               symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double         iBearsPowerBuffer[];
//--- 存储 iBearsPower 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在熊市指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iBearsPowerBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iBearsPower)
      handle=iBearsPower(name,period,ma_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- ma周期
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BEARS,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iBearsPower indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示熊市指标计算的交易品种/时间帧
   short_name=StringFormat("iBearsPower(%s/%s, period=%d)",name,EnumToString(period),r
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iBearsPower指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iBearsPower 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iBearsPowerBuffer数组大于交易品种/周期 iBearsPower 指标的数量值，那么我们不会
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以熊市指标的值填充iBearsPowerBuffer 数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iBearsPowerBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住熊市指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iBearsPower 指标的指标缓冲区                                  |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 熊市值的指标缓冲区
                         int ind_handle,    // iBearsPower 指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iBearsPowerBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iBearsPower indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                  |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iBands

函数返回布林带指标处理器。

```
int  iBands(
   string               symbol,           // 交易品种名称
   ENUM_TIMEFRAMES      period,           // 周期
   int                  bands_period,     // 平均线计算周期
   int                  bands_shift,      // 指标平移
   double               deviation,        // 标准差数
   ENUM_APPLIED_PRICE   applied_price     // 价格或处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*bands_period*

[in] 指标主线的平均周期。

*bands_shift*

[in] 相关价格图表的指标转换。

*deviation*

[in] 偏离主线。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0 - BASE_LINE, 1 - UPPER_BAND, 2 - LOW ER_BAND。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iBands.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "of indicator buffers for the iBands technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- 上标图
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 下标图
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrMediumSeaGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 中间标图
#property indicator_label3  "Middle"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrMediumSeaGreen
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBands,             // 使用iBands
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iBands;          // 函数的类型
input int                   bands_period=20;           // 平均移动周期
input int                   bands_shift=0;             // 移动
input double                deviation=2.0;             //标准偏差数
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
double         UpperBuffer[];
double         LowerBuffer[];
double         MiddleBuffer[];
//--- 存储 iBands 指标处理程序的变量
int    handle;
//--- 存储变量
```

© 2000-2025, MetaQuotes Ltd.

```
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在布林带指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,MiddleBuffer,INDICATOR_DATA);
//--- 设置每条线的移动
   PlotIndexSetInteger(0,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,bands_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iBands)
      handle=iBands(name,period,bands_period,bands_shift,deviation,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- ma周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=bands_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=bands_shift;
      //--- 标准偏差数
      pars[2].type=TYPE_DOUBLE;
      pars[2].double_value=deviation;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_BANDS,4,pars);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iBands indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示布林带指标计算的交易品种/时间帧
   short_name=StringFormat("iBands(%s/%s, %d,%d,%G,%s)",name,EnumToString(period),
                           bands_period,bands_shift,deviation,EnumToString(applied_pri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iBands 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iBands 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果指标缓冲区的大小大于交易品种/周期iBands 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
```

© 2000-2025, MetaQuotes Ltd.

```
      else                          values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以布林带指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(MiddleBuffer,UpperBuffer,LowerBuffer,bands_shift,handle,v
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住布林带指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iBands指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &base_values[],     // 布林带中间线的指标缓冲区
                           double &upper_values[],    // 上边界的指标缓冲区
                           double &lower_values[],    // 下边界的指标缓冲区
                           int shift,                 // 移动
                           int ind_handle,            // iBands 指标的处理程序
                           int amount                 // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分MiddleBuffer数组
   if(CopyBuffer(ind_handle,0,-shift,amount,base_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分UpperBuffer 数组
   if(CopyBuffer(ind_handle,1,-shift,amount,upper_values)<0)
     {
      //--- 如果复制失败，显示错误代码
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分 LowerBuffer 数组
   if(CopyBuffer(ind_handle,2,-shift,amount,lower_values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iBullsPower

函数返回牛市指标处理器。只有一个缓冲区。

```
int  iBullsPower(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算平均周期值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iBullsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBullsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iBullsPower标图
#property indicator_label1  "iBullsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBullsPower,       // 使用iBullsPower
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iBullsPower;  // 函数类型
input int                  ma_period=13;           // 平均移动周期
input string               symbol=" ";             // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // 时间帧
//--- 指标缓冲区
double         iBullsPowerBuffer[];
//--- 存储 iBullsPower指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在牛市指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iBullsPowerBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iBullsPower)
      handle=iBullsPower(name,period,ma_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- ma周期
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BULLS,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iBullsPower indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示牛市指标计算的交易品种/时间帧
   short_name=StringFormat("iBullsPower(%s/%s, period=%d)",name,EnumToString(period),r
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iBullsPower 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iBullsPower 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iBullsPowerBuffer 数组大于交易品种/周期 iBullsPower 指标的数量值，那么我们不会
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以牛市指标的值填充iBullsPowerBuffer 数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iBullsPowerBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住牛市指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iBullsPower 指标的指标缓冲区                                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 牛市值的指标缓冲区
                         int ind_handle,    // iBullsPower 指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iBullsPowerBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iBullsPower indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# iCCI

函数返回顺势指标处理器。只有一个缓冲区。

```
int  iCCI(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   ENUM_APPLIED_PRICE   applied_price      // 价格或处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算平均周期值。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                    Demo_iCCI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iCCI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iCCI 标图
#property indicator_label1  "iCCI"
#property indicator_type1   DRAW_LINE
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标窗口的横向水平
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iCCI,              // 使用iCCI
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iCCI;               // 函数类型
input int                  ma_period=14;                 // 平均移动周期
input ENUM_APPLIED_PRICE   applied_price=PRICE_TYPICAL;  // 价格类型
input string               symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double         iCCIBuffer[];
//--- 存储 iCCI 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在商品通道指数指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iCCIBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
```

© 2000-2025, MetaQuotes Ltd.

```
   if(type==Call_iCCI)
      handle=iCCI(name,period,ma_period,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[2];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 价格类型
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_CCI,2,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iCCI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示布林带® 指标计算的交易品种/时间帧
   short_name=StringFormat("iCCI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iCCI 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
```

© 2000-2025, MetaQuotes Ltd.

```
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iCCI指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iCCIBuffer 数组大于交易品种/周期 iCCI 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以商品通道指数指标的值填充iCCIBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iCCIBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住商品通道指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iCCI指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 商品通道指数值的指标缓冲区
                         int ind_handle,    // iCCI 指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iCCIBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iCCI indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iChaikin

函数返回蔡金摆动指标的处理程序。它只有一个缓冲区。

```
int  iChaikin(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  fast_ma_period,      // 快速周期
   int                  slow_ma_period,      // 慢速周期
   ENUM_MA_METHOD       ma_method,           // 平滑类型
   ENUM_APPLIED_VOLUME  applied_volume       // 交易量类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*fast_ma_period*

[in] 计算快速平均周期。

*slow_ma_period*

[in] 计算慢速平均周期。

*ma_method*

[in] 平滑类型。可以是 ENUM_MA_METHOD平均常量中的一个。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 常量中的一个。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iChaikin.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iChaikin technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iChaikin标图
#property indicator_label1  "iChaikin"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iChaikin,           // 使用iChaikin
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iChaikin;           // 函数类型
input int                   fast_ma_period=3;             // 快速ma周期
input int                   slow_ma_period=10;            // 慢速ma周期
input ENUM_MA_METHOD        ma_method=MODE_EMA;           // 平滑类型
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // 交易量类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          iChaikinBuffer[];
//--- 存储 iChaikin 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在蔡金摆动指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iChaikinBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iChaikin)
      handle=iChaikin(name,period,fast_ma_period,slow_ma_period,ma_method,applied_volu
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- 快速ma周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ma_period;
      //--- 慢速ma周期
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ma_period;
      //--- 平滑类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- 交易量类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_CHAIKIN,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iChaikin indicator for the symbol %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示蔡金摆动指标计算的交易品种/时间帧
   short_name=StringFormat("iChaikin(%s/%s, %d, %d, %s, %s)",name,EnumToString(period)
                           fast_ma_period,slow_ma_period,
                           EnumToString(ma_method),EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iChaikin 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iChaikin 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iChaikinBuffer 数组大于交易品种/周期iChaikin 指标的数量值，那么我们不会复制任
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以蔡金摆动指标的值填充iChaikinBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iChaikinBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住蔡金摆动指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 填充 iChaikin 指标的指标缓冲区                                   |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 蔡金摆动指标值的指标缓冲区
                         int ind_handle,    // iChaikin 指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//---以0标引指标缓冲区的值填充部分iChaikinBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iChaikin indicator, error code %d",Get
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iCustom

函数返回指定自定义指标的处理器。

```
int  iCustom(
   string           symbol,     // 交易品种名称
   ENUM_TIMEFRAMES  period,     // 周期
   string           name        // 文件夹/自定义指标_名称
   ...                          // 指标输入参量列表
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*name*

[in] 如果在自定义指标名称前指出反斜杠符号'\'，则相对于MQL5根文件夹搜索EX5指标文件。因此，对于调用iCustom(Symbol()、Period()、"\FirstIndicator"...)，该指标将被加载为MQL5\FirstIndicator.ex5。如果在此路径中找不到此文件，则返回错误error 4802 (ERR_INDICATOR_CANNOT_CREATE)。

如果路径不是以'\'开始，则会根据以下方式搜索和下载指标：

- 首先，在调用程序的EX5文件所在的文件夹中搜索指标EX5文件。例如，CrossMA.EX5 EA位于MQL5\Experts\MyExperts，并包含iCustom调用(Symbol()、Period()、"SecondIndicator"...)。在这种情况下，将在MQL5\Experts\MyExperts\SecondIndicator.ex5中搜索指标。
- 如果在同目录下没有找到指标，则相对于MQL5\Indicators指标根目录执行搜索。换句话说，将执行对MQL5\Indicators\SecondIndicator.ex5文件的搜索。如果找不到指标，则该函数返回INVALID_HANDLE并触发错误4802(ERR_INDICATOR_CANNOT_CREATE)。

如果在子目录（例如MyIndicators\ThirdIndicator)中设置指标路径，则首先在MQL5\Experts\MyExperts\MyIndicators\ThirdIndicator.ex5的调用程序文件夹（EA交易位于MQL5\Experts\MyExperts)中执行搜索。如果未成功，则执行对MQL5\Indicators\MyIndicators\ThirdIndicator.ex5文件的搜索。确保在路径中使用双反斜杠'\\'作为分隔符，例如iCustom (Symbol()、Period()、"MyIndicators\\ThirdIndicator"...)

*...*

[in] 自定义指标的 输入-参量 ，逗号分开，类型和参量命令必须匹配，如果没有指定参量，使用 默认值 。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

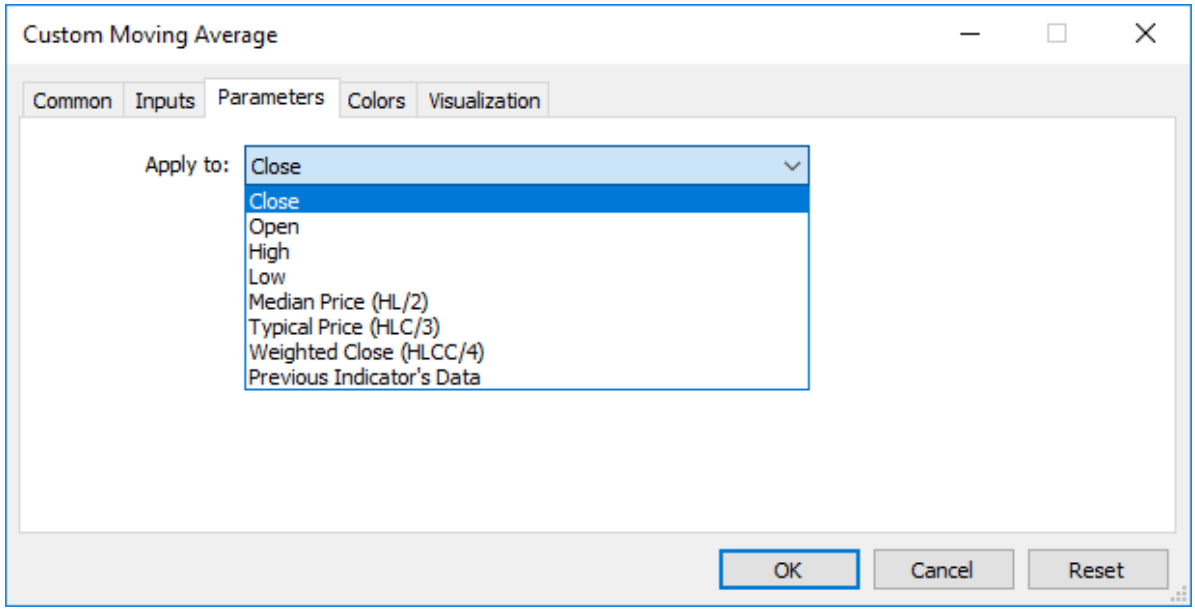
必须编辑自定义指标（扩展EX5)并且在可获端或子目录中保存在MQL5/Indicators目录中。

© 2000-2025, MetaQuotes Ltd.

需要测试的指标是指自动从iCustom()函数调用，如果相应的参数是通过一个通用的常量字符串集合。对于其它情况下（使用IndicatorCreate()函数或一个非常量字符串设定指标名称）该属性#property tester_indicator是必需的：

```
#property tester_indicator "indicator_name.ex5"
```

如果在指标中使用第一调用方式，然后在自定义指标中可以在"Parameters"标签中自动形成为指示数据的计算，如果"Apply to"参量不是明确指定的，默认计算将以"Close"价格值为主。




当从MQL5程序中调用自定义指标时，Applied_Price参量或者另外的指标处理器应该最后传递，在所有自定义指标的输入变量之后。

另见

程序属性，时间序列和指标访问，IndicatorCreate()，IndicatorRelease()

示例：

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- 标签图1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 输入参量
input int MA_Period=21;
input int MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMA;
//--- 指标缓冲区
double         Label1Buffer[];
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 自定义移动平均数处理程序。mq5自定义指标
int MA_handle;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 指标缓冲区绘图
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
   ResetLastError();
   MA_handle=iCustom(NULL,0,"Examples\\Custom Moving Average",
                     MA_Period,
                     MA_Shift,
                     MA_Method,
                     PRICE_CLOSE // 使用收盘价
                     );
   Print("MA_handle = ",MA_handle,"  error = ",GetLastError());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标重复函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 复制指标自定义移动平均值到指标缓冲区
   int copy=CopyBuffer(MA_handle,0,0,rates_total,Label1Buffer);
   Print("copy = ",copy,"    rates_total = ",rates_total);
//--- 如果尝试失败-这里报道
   if(copy<=0)
      Print("An attempt to get the values if Custom Moving Average has failed");
//--- 为下次调用返回prev_calculated值
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

# iDEMA

函数返回双指数移动平均线指标处理器。只有一个缓冲区。

```
int  iDEMA(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   int                  ma_shift,          // 平移
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 计算平均周期（计算柱）。

*ma_shift*

[in] 关于价格图表转变的指标。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iDEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_plots   1
//--- iDEMA 标图
#property indicator_label1  "iDEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDEMA,              // 使用iDEMA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iDEMA;            // 函数类型
input int                   ma_period=14;               // 平均移动周期
input int                   ma_shift=0;                 // 移动
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;  // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iDEMABuffer[];
//--- 存储iDEMA 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在双指数移动平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iDEMABuffer,INDICATOR_DATA);
   //--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iDEMA)
      handle=iDEMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[3];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- 价格类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_DEMA,3,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iDEMA indicator for the symbol %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示双指数移动平均指标计算的交易品种/时间帧
   short_name=StringFormat("iDEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iDEMA 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iDEMA指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iDEMABuffer 数组大于交易品种/周期 iDEMA 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以双指数移动平均指标的值填充iDEMABuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iDEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住双指数移动平均指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iDEMA指标的指标缓冲区                                          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // 双指数移动平均值的指标缓冲区
                         int shift,          // 移动
                         int ind_handle,     // iDEMA指标的处理程序
```

© 2000-2025, MetaQuotes Ltd.

```
                           int amount          // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iDEMABuffer 数组
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iDEMA indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iDeMarker

函数返回DeMarker指标处理器。只有一个缓冲区。

```
int  iDeMarker(
   string           symbol,        // 交易品种名称
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 计算平均周期（计算柱）。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iDeMarker.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDeMarker technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iDeMarker 标图
#property indicator_label1  "iDeMarker"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 指标窗口的横向水平
#property indicator_level1  0.3
#property indicator_level2  0.7
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDeMarker,          // 使用iDeMarker
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iDeMarker;       // 函数类型
input int                   ma_period=14;              // 平均移动周期
input string                symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
double          iDeMarkerBuffer[];
//--- 存储iDeMarker指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在DeMarker 指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iDeMarkerBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iDeMarker)
      handle=iDeMarker(name,period,ma_period);
   else
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_DEMARKER,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iDeMarker indicator for the symbol %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示DeMarker 指标计算的交易品种/时间帧
   short_name=StringFormat("iDeMarker(%s/%s, period=%d)",name,EnumToString(period),ma_
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iDeMarker指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iDeMarker 指标数量值更改
//---或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
```

© 2000-2025, MetaQuotes Ltd.

```
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iDeMarkerBuffer数组大于交易品种/周期iDeMarker 指标的数量值，那么我们不会复制任
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以DeMarker 指标的值填充iDeMarkerBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iDeMarkerBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住DeMarker 指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iDeMarker 指标的指标缓冲区                                   |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // DeMarker值的指标缓冲区
                         int ind_handle,     // iDeMarker 指标的处理程序
                         int amount          // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iDeMarkerBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iDeMarker indicator, error code %d",Ge
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iEnvelopes

函数返回轨道线指标处理器。

```
int  iEnvelopes(
   string               symbol,           // 交易品种名称
   ENUM_TIMEFRAMES      period,           // 周期
   int                  ma_period,        // 平均线计算周期
   int                  ma_shift,         // 指标平移
   ENUM_MA_METHOD       ma_method,        // 平滑类型
   ENUM_APPLIED_PRICE   applied_price,    // 价格或者处理器类型
   double               deviation         // 中线边界差(百分率)
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 主线平均周期

*ma_shift*

[in] 相关价格图表的指标转换。

*ma_method*

[in] 平滑类型。可以是 ENUM_MA_METHOD 值中的一个。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

*deviation*

[in] 偏离主线 (百分比)。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码：0 -UPPER_LINE，1 -LOWER_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iEnvelopes.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iEnvelopes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- 上标图
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 下标图
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iEnvelopes,         // 使用iEnvelopes
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iEnvelopes;      // 函数类型
input int                  ma_period=14;              // 平均移动周期
input int                  ma_shift=0;                // 移动
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // 平滑类型
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // 价格类型
input double               deviation=0.1;             // 移动平均的边界偏差
input string               symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
double         UpperBuffer[];
double         LowerBuffer[];
//--- 存储iEnvelopes指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
```

© 2000-2025, MetaQuotes Ltd.

```
string short_name;
//--- 我们将在轨道线指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
//--- 设置每条线的移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iEnvelopes)
      handle=iEnvelopes(name,period,ma_period,ma_shift,ma_method,applied_price,deviati
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[5];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- 平滑类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      //--- 价格类型
      pars[4].type=TYPE_DOUBLE;
      pars[4].double_value=deviation;
      handle=IndicatorCreate(name,period,IND_ENVELOPES,5,pars);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iEnvelopes indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示轨道线指标计算的交易品种/时间帧
   short_name=StringFormat("iEnvelopes(%s/%s, %d, %d, %s,%s, %G)",name,EnumToString(pe
   ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price),deviation);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iEnvelopes指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iEnvelopes指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果UpperBuffer 数组大于交易品种/周期iEnvelopes 指标的数量值，那么我们不会复制任何
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   else
      {
       //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
       //--- 为了计算，添加不超过一柱
       values_to_copy=(rates_total-prev_calculated)+1;
      }
//--- 用来自轨道线指标的值填充UpperBuffer和LowerBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(UpperBuffer,LowerBuffer,ma_shift,handle,values_to_copy))
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住轨道线指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iEnvelopes指标的指标缓冲区                                   |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &upper_values[],     // 上边界的指标缓冲区
                           double &lower_values[],     // 下边界的指标缓冲区
                           int shift,                  // 移动
                           int ind_handle,             // 轨道线指标的处理程序
                           int amount                  // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分UpperBuffer 数组
   if(CopyBuffer(ind_handle,0,-shift,amount,upper_values)<0)
      {
       //--- 如果复制失败，显示错误代码
       PrintFormat("Failed to copy data from the iEnvelopes indicator, error code %d",G
       //--- 退出零结果 - 它表示被认为是不计算的指标
       return(false);
      }
//--- 以1标引指标缓冲区的值填充部分LowerBuffer数组
   if(CopyBuffer(ind_handle,1,-shift,amount,lower_values)<0)
      {
       //--- 如果复制失败，显示错误代码
       PrintFormat("Failed to copy data from the iEnvelopes indicator, error code %d",G
       //--- 退出零结果 - 它表示被认为是不计算的指标
       return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iForce

函数返回强力指数指标处理器。只有一个缓冲区。

```
int  iForce(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   ENUM_MA_METHOD       ma_method,         // 平滑类型
   ENUM_APPLIED_VOLUME  applied_volume     // 计算的交易量类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算的平均周期。

*ma_method*

[in] 平滑类型。可以是 ENUM_MA_METHOD 值中的一个。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 值中的一个。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iForce.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iForce technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_plots   1
//--- 绘制iForce
#property indicator_label1  "iForce"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iForce,             // 使用iForce
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iForce;             // 函数类型
input int                   ma_period=13;                 // 平均移动周期
input ENUM_MA_METHOD        ma_method=MODE_SMA;           // 平滑类型
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // 交易量类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          iForceBuffer[];
//--- 存储iForce指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在强力指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iForceBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 创建指标处理程序
   if(type==Call_iForce)
      handle=iForce(name,period,ma_period,ma_method,applied_volume);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[3];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 平滑类型
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_method;
      //--- 交易量类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_volume;
      //--- 价格类型
      handle=IndicatorCreate(name,period,IND_FORCE,3,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iForce indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示强力指标计算的交易品种/时间帧
   short_name=StringFormat("iForce(%s/%s, %d, %s, %s)",name,EnumToString(period),
                           ma_period,EnumToString(ma_method),EnumToString(applied_volu
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const long &volume[],
                const int &spread[])
  {
//--- 从iForce指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iForce指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iForceBuffer 数组大于交易品种/周期iForce 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以强力指标的值填充iForceBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iForceBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住强力指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iForce指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // 强力指数值的指标缓冲区
                         int ind_handle,     // iForce指标的处理程序
                         int amount          // 复制值的数量
                         )
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iForceBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iForce indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iFractals

函数返回分形学指标处理器。

```
int  iFractals(
   string           symbol,     // 交易品种名称
   ENUM_TIMEFRAMES  period      // 周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0 - UPPER_LINE，1 - LOWER_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iFractals.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFractals technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- FractalUp 标图
#property indicator_label1  "FractalUp"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
```

© 2000-2025, MetaQuotes Ltd.

```
//--- FractalDown 标图
#property indicator_label2  "FractalDown"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFractals,          // 使用iFractals
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iFractals;          // 函数类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          FractalUpBuffer[];
double          FractalDownBuffer[];
//--- 存储iFractals指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在分形学指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,FractalUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,FractalDownBuffer,INDICATOR_DATA);
//--- 使用来自Wingdings字符集的交易品种为PLOT_ARROW属性设置代码
   PlotIndexSetInteger(0,PLOT_ARROW,217); // 箭头向上
   PlotIndexSetInteger(1,PLOT_ARROW,218); // 箭头向下
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建指标处理程序
   if(type==Call_iFractals)
      handle=iFractals(name,period);
   else
      handle=IndicatorCreate(name,period,IND_FRACTALS);
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iFractals indicator for the symbol %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示分形学指标计算的交易品种/时间帧
   short_name=StringFormat("iFractals(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iFractals指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iFractals指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果FractalUpBuffer数组大于交易品种/周期iFractals指标的数量值，那么我们不会复制任何
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 用来自分形学指标的值填充FractalUpBuffer和FractalDownBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(FractalUpBuffer,FractalDownBuffer,handle,values_to_copy))
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住分形学指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iFractals指标的指标缓冲区                                    |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &up_arrows[],         // 向上箭头的指标缓冲区
                           double &down_arrows[],       // 向下箭头的指标缓冲区
                           int ind_handle,              // iFractals指标的处理程序
                           int amount                   // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分FractalUpBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,up_arrows)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iFractals indicator to the FractalUpBu
                  GetLastError());
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 以1标引指标缓冲区的值填充部分FractalDownBuffer数组
   if(CopyBuffer(ind_handle,1,0,amount,down_arrows)<0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iFractals indicator to the FractalDown
                  GetLastError());
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iFrAMA

函数返回分形学适合移动平均指标处理器。只有一个缓冲区。

```
int  iFrAMA(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   int                  ma_shift,          // 图表平移
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算周期（计算柱）。

*ma_shift*

[in] 在价格图表中的指标转换。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iFrAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFrAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_plots   1
//--- 绘制iFrAMA
#property indicator_label1  "iFrAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFrAMA,             // 使用iFrAMA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iFrAMA;              // 函数类型
input int                   ma_period=14;                  // 平均移动周期
input int                   ma_shift=0;                    // 移动
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;     // 价格类型
input string                symbol=" ";                    // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;         // 时间帧
//--- 指标缓冲区
double          iFrAMABuffer[];
//--- 存储iFrAMA 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在分形学适合移动平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iFrAMABuffer,INDICATOR_DATA);
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iFrAMA)
      handle=iFrAMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[3];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- 价格类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      //--- 价格类型
      handle=IndicatorCreate(name,period,IND_FRAMA,3,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iFrAMA indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示iFrAMA indicator指标计算的交易品种/时间帧
   short_name=StringFormat("iFrAMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iFrAMA指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iFrAMA指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iFrAMABuffer数组大于交易品种/周期iFrAMA 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以分形学适合移动平均指标的值填充iFrAMABuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iFrAMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住分形学适合移动平均指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iFrAMA指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 分形学适合移动平均值的指标缓冲区
                         int shift,         // 移动
```

© 2000-2025, MetaQuotes Ltd.

```
                           int ind_handle,     // iFrAMA 指标的处理程序
                           int amount          // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iFrAMABuffer数组
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iFrAMA indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iGator

函数返回鳄鱼振荡器指标处理器。振荡器表示鳄鱼指标蓝线和红线（上升柱状图）的区别以及红线和绿线的区别（下降柱状图）。

```
int  iGator(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  jaw_period,          // 咽喉计算周期
   int                  jaw_shift,           // 咽喉平移
   int                  teeth_period,        // 牙齿计算周期
   int                  teeth_shift,         // 牙齿平移
   int                  lips_period,         // 唇部计算周期
   int                  lips_shift,          // 唇部平移
   ENUM_MA_METHOD       ma_method,           // 平滑类型
   ENUM_APPLIED_PRICE   applied_price        // 价格或者处理器类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL值代表当前交易品种。

*period*

[in]周期值可以是ENUM_TIMEFRAMES值中的一个，0代表当前时间表。

*jaw_period*

[in]蓝线平均周期（鳄鱼颌骨）。

*jaw_shift*

[in]与价格图表有关的蓝线变化。不能直接连接到指标柱状图的视觉变化。

*teeth_period*

[in]红线的平均周期（鳄鱼牙）。

*teeth_shift*

[in]关于价格图表的红线变化。不能直接连接到指标柱状图的视觉变化。

*lips_period*

[in]绿线的平均周期（鳄鱼唇）。

*lips_shift*

[in]关于价格图表的绿线变化。不能直接连接到指标柱状图的视觉变化。

*ma_method*

[in]平滑类型。可以是ENUM_MA_METHOD值中的一个。

*applied_price*

[in]使用价格。可以是任意ENUM_APPLIED_PRICE价格常量或者另外指标处理器。

返回值

© 2000-2025, MetaQuotes Ltd.

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码：0-UPPER_HISTOGRAM，1-上升柱状图的颜色缓冲区，2 - LOWER_HISTOGRAM，3 - 下降柱状图的颜色缓冲区。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iGator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iGator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters are as in a standard Gator Oscillator."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
//--- 绘制GatorUp
#property indicator_label1  "GatorUp"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 绘制GatorDown
#property indicator_label2  "GatorDown"
#property indicator_type2   DRAW_COLOR_HISTOGRAM
#property indicator_color2  clrGreen, clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iGator,            // 使用iGator
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iGator;       // 函数类型
```

© 2000-2025, MetaQuotes Ltd.

```
input string                symbol=" ";              // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // 时间帧
input int                   jaw_period=13;           // 咽喉线周期
input int                   jaw_shift=8;             // 咽喉线移动
input int                   teeth_period=8;          // 牙齿线周期
input int                   teeth_shift=5;           // 牙齿线移动
input int                   lips_period=5;           // 唇部线周期
input int                   lips_shift=3;            // 唇部线移动
input ENUM_MA_METHOD        MA_method=MODE_SMMA;     // 鳄鱼平均线的方法
input ENUM_APPLIED_PRICE    applied_price=PRICE_MEDIAN;// 用于计算鳄鱼的价格类型
//--- 指标缓冲区
double          GatorUpBuffer[];
double          GatorUpColors[];
double          GatorDownBuffer[];
double          GatorDownColors[];
//--- 存储iGator 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 移动上下直方图的值
int shift;
//--- 我们将在加多摆动指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,GatorUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,GatorUpColors,INDICATOR_COLOR_INDEX);
   SetIndexBuffer(2,GatorDownBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,GatorDownColors,INDICATOR_COLOR_INDEX);
/*
  参数中指定的所有移动指的是绘制加多摆动指标基础上的鳄鱼指标!
  这就是我们不能移动加多摆动指标本身的原因,但是他们移动鳄鱼线,
  其值用于计算加多摆动指标!
*/
//--- 我们计算上下直方图的移动,这等于咽喉线和牙齿线之间的区别
   shift=MathMin(jaw_shift,teeth_shift);
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//--- 尽管指标包含两个直方图,也使用相同的移动 - 就是实施iGator 指标。
   PlotIndexSetInteger(1,PLOT_SHIFT,shift);

//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
```

© 2000-2025, MetaQuotes Ltd.

```
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iGator)
      handle=iGator(name,period,jaw_period,jaw_shift,teeth_period,teeth_shift,
                    lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[8];
      //--- 鳄鱼线的周期和移动
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
      //--- 平滑类型
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
      //--- 价格类型
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
      //--- 创建处理程序
      handle=IndicatorCreate(name,period,IND_GATOR,8,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iGator indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 显示加多摆动指标计算的交易品种/时间帧
   short_name=StringFormat("iGator(%s/%s, %d, %d ,%d, %d, %d, %d)",name,EnumToString(p
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,l
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iGator指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iGator指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果GatorUpBuffer数组大于交易品种/周期iGator指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以加多摆动指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(GatorUpBuffer,GatorUpColors,GatorDownBuffer,GatorDownColo
      shift,handle,values_to_copy)) return(0);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住加多摆动指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iGator 指标的指标缓冲区                                     |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &ups_buffer[],          // 向上直方图的指标缓冲区
                           double &up_color_buffer[],     // 向上直方图价格指数的指标缓冲区
                           double &downs_buffer[],        // 向下直方图的指标缓冲区
                           double &downs_color_buffer[], // 向下直方图价格指数的指标缓冲区
                           int u_shift,                   // 上下直方图移动
                           int ind_handle,                // iGator 指标的处理程序
                           int amount                     // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分GatorUpBuffer 数组
   if(CopyBuffer(ind_handle,0,-u_shift,amount,ups_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分GatorUpColors 数组
   if(CopyBuffer(ind_handle,1,-u_shift,amount,up_color_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分GatorDownBuffer 数组
   if(CopyBuffer(ind_handle,2,-u_shift,amount,downs_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以3标引指标缓冲区的值填充部分GatorDownColors 数组
   if(CopyBuffer(ind_handle,3,-u_shift,amount,downs_color_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iIchimoku

函数返回一目均衡图指标处理器。

```
int  iIchimoku(
   string           symbol,           // 交易品种类型
   ENUM_TIMEFRAMES  period,           // 周期
   int              tenkan_sen,       // Tenkan-sen转换线周期
   int              kijun_sen,        // Kijun-sen基准线周期
   int              senkou_span_b     // Senkou Span B周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*tenkan_sen*

[in] 转换线平均周期。

*kijun_sen*

[in] 基准线平均周期。

*senkou_span_b*

[in] Senkou Span B 平均周期。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码：0 - TENKANSEN_LINE, 1 - KIJUNSEN_LINE, 2 - SENKOUSPANA_LINE, 3 - SENKOUSPANB_LINE, 4 - CHIKOUSPAN_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iIchimoku.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iIchimoku technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters just like in the standard Ichimoku Kinko H

#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   4
//--- Tenkan_sen 标图
#property indicator_label1  "Tenkan_sen"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- Kijun_sen 标图
#property indicator_label2  "Kijun_sen"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrBlue
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- Senkou_Span 标图
#property indicator_label3  "Senkou Span A;Senkou Span B" // two fields will be shown
#property indicator_type3   DRAW_FILLING
#property indicator_color3  clrSandyBrown, clrThistle
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//--- Chinkou_Span 标图
#property indicator_label4  "Chinkou_Span"
#property indicator_type4   DRAW_LINE
#property indicator_color4  clrLime
#property indicator_style4  STYLE_SOLID
#property indicator_width4  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iIchimoku,         // 使用iIchimoku
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iIchimoku;       // 函数类型
input int                  tenkan_sen=9;              // Tenkan-sen周期
input int                  kijun_sen=26;              // Kijun-sen周期
input int                  senkou_span_b=52;          // Senkou Span B周期
input string               symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
double         Tenkan_sen_Buffer[];
double         Kijun_sen_Buffer[];
double         Senkou_Span_A_Buffer[];
```

© 2000-2025, MetaQuotes Ltd.

```
double         Senkou_Span_B_Buffer[];
double         Chinkou_Span_Buffer[];
//--- 存储iIchimoku指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在一目均衡图指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,Tenkan_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(1,Kijun_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(2,Senkou_Span_A_Buffer,INDICATOR_DATA);
   SetIndexBuffer(3,Senkou_Span_B_Buffer,INDICATOR_DATA);
   SetIndexBuffer(4,Chinkou_Span_Buffer,INDICATOR_DATA);
//--- 设置未来方向kijun_sen 柱形的Senkou 跨通道的移动
   PlotIndexSetInteger(2,PLOT_SHIFT,kijun_sen);
//--- 设置无需Chinkou跨线的移动，因为数据跨度
//--- 已经存储在iIchimoku移动
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iIchimoku)
      handle=iIchimoku(name,period,tenkan_sen,kijun_sen,senkou_span_b);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[3];
      //--- 鳄鱼线的周期和移动
      pars[0].type=TYPE_INT;
      pars[0].integer_value=tenkan_sen;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=kijun_sen;
      pars[2].type=TYPE_INT;
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[2].integer_value=senkou_span_b;
      //--- 创建处理程序
      handle=IndicatorCreate(name,period,IND_ICHIMOKU,3,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iIchimoku indicator for the symbol %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示一目均衡图指标计算的交易品种/时间帧
   short_name=StringFormat("iIchimoku(%s/%s, %d, %d ,%d)",name,EnumToString(period),
                           tenkan_sen,kijun_sen,senkou_span_b);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iIchimoku指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iIchimoku指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果Tenkan_sen_Buffer数组大于交易品种/周期iIchimoku指标的数量值，那么我们不会复制
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以一目均衡图指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(Tenkan_sen_Buffer,Kijun_sen_Buffer,Senkou_Span_A_Buffer,S
      kijun_sen,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住一目均衡图指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iIchimoku 指标的指标缓冲区                                  |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &tenkan_sen_buffer[],     // Tenkan-sen线的指标缓冲区
                           double &kijun_sen_buffer[],      // Kijun_sen线的指标缓冲区
                           double &senkou_span_A_buffer[],  // Senkou Span A 线的指标缓
                           double &senkou_span_B_buffer[],  // Senkou Span B线的指标缓冲
                           double &chinkou_span_buffer[],   // Chinkou Span线的指标缓冲
                           int senkou_span_shift,           // 未来方向Senkou 的跨线移动
                           int ind_handle,                  // iIchimoku 指标的处理程序
                           int amount                       // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分Tenkan_sen_Buffer数组
   if(CopyBuffer(ind_handle,0,0,amount,tenkan_sen_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("1.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- 以1标引指标缓冲区的值填充部分Kijun_sen_Buffer 数组
   if(CopyBuffer(ind_handle,1,0,amount,kijun_sen_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("2.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以2标引指标缓冲区的值填充部分Chinkou_Span_Buffer 数组
//--- 如果senkou_span_shift>0，线根据senkou_span_shift柱形的未来方向移动。
   if(CopyBuffer(ind_handle,2,-senkou_span_shift,amount,senkou_span_A_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("3.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以3标引指标缓冲区的值填充部分Senkou_Span_A_Buffer 数组
//--- 如果senkou_span_shift>0，线根据senkou_span_shift柱形的未来方向移动。
   if(CopyBuffer(ind_handle,3,-senkou_span_shift,amount,senkou_span_B_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("4.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以0标引指标缓冲区的值填充部分Senkou_Span_B_Buffer 数组
//--- 当复制Chinkou Span时，我们无需考虑移动，因为Chinkou数据跨度
//--- 已经存储在iIchimoku移动
   if(CopyBuffer(ind_handle,4,0,amount,chinkou_span_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("5.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
    if(handle!=INVALID_HANDLE)
       IndicatorRelease(handle);
//--- 删除指标后清空图表
    Comment("");
   }
```

© 2000-2025, MetaQuotes Ltd.

# iBWMFI

函数返回市场便利指标处理器。只有一个缓冲区。

```
int  iBWMFI(
   string               symbol,              // 交易品种类型
   ENUM_TIMEFRAMES      period,              // 周期
   ENUM_APPLIED_VOLUME  applied_volume       // 计算的交易量类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 常量中的一个。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iBWMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBWMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- iBWMFI 标图
#property indicator_label1  "iBWMFI"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrLime,clrSaddleBrown,clrBlue,clrPink
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBWMFI,             // 使用iBWMFI
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iBWMFI;           // 函数类型
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;// 交易量类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iBWMFIBuffer[];
double          iBWMFIColors[];
//--- 存储 iBWMFI 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在市场便利指数比尔 威廉姆斯指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iBWMFIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iBWMFIColors,INDICATOR_COLOR_INDEX);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iBWMFI)
      handle=iBWMFI(name,period,applied_volume);
   else
     {
      //--- 以指标参数填充结构
```

© 2000-2025, MetaQuotes Ltd.

```
      MqlParam pars[1];
      //--- 交易量类型
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_BWMFI,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iBWMFI indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示市场便利指数比尔 威廉姆斯指标计算的交易品种/时间帧
   short_name=StringFormat("iBWMFI(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iBWMFI 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iBWMFI指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
```

© 2000-2025, MetaQuotes Ltd.

```
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iBWMFIBuffer 数组大于交易品种/周期 iBWMFI 指标的数量值，那么我们不会复制任何内
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以市场便利指数比尔 威廉姆斯指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(iBWMFIBuffer,iBWMFIColors,handle,values_to_copy)) return
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住市场便利指数比尔 威廉姆斯指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iBWMFI 指标的指标缓冲区                                      |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &values[],     // 直方图值的指标缓冲区
                           double &colors[],     // 直方图颜色的指标缓冲区
                           int ind_handle,       // iBWMFI 指标的处理程序
                           int amount            // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iBWMFIBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//---以1标引指标缓冲区的值填充部分 iBWMFIColors 数组
   if(CopyBuffer(ind_handle,1,0,amount,colors)<0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 如果复制失败，显示错误代码
       PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLa
       //--- 退出零结果 - 它表示被认为是不计算的指标
       return(false);
      }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iMomentum

函数返回动量指标处理器。只有一个缓冲区。

```
int  iMomentum(
   string                symbol,              // 交易品种名称
   ENUM_TIMEFRAMES       period,              // 周期
   int                   mom_period,          // 平均周期
   ENUM_APPLIED_PRICE    applied_price        // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*mom_period*

[in] 计算价格变化的平均周期（计算柱）。

*applied_price*

[in] 使用价格，可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iMomentum.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMomentum technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Momentum."

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 标图iMomentum
#property indicator_label1  "iMomentum"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMomentum,          // 使用iMomentum
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iMomentum;        // 函数类型
input int                   mom_period=14;              // 动量周期
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iMomentumBuffer[];
//--- 存储 iMomentum 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在动量指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iMomentumBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iMomentum)
      handle=iMomentum(name,period,mom_period,applied_price);
```

© 2000-2025, MetaQuotes Ltd.

```
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[2];
      //--- 周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=mom_period;
      //--- 价格类型
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MOMENTUM,2,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iMomentum indicator for the symbol %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示动量指标计算的交易品种/时间帧
   short_name=StringFormat("iMomentum(%s/%s, %d, %s)",name,EnumToString(period),
                           mom_period, EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iMomentum指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
       return(0);
      }
//--- 如果它是指标计算的最初起点或如果 iMomentum指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
      {
       //--- 如果iMomentumBuffer数组大于交易品种/周期iMomentum指标的数量值，那么我们不会复制任何
       //--- 否则，我们复制小于指标缓冲区的大小
       if(calculated>rates_total) values_to_copy=rates_total;
       else                       values_to_copy=calculated;
      }
   else
      {
       //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
       //--- 为了计算，添加不超过一柱
       values_to_copy=(rates_total-prev_calculated)+1;
      }
//--- 以动量指标的值填充iMomentumBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iMomentumBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住动量指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iMomentum指标的指标缓冲区                                   |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // 动量值的指标缓冲区
                         int ind_handle,    // iMomentum 指标的处理程序
                         int amount         // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iMomentumBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
      {
       //--- 如果复制失败，显示错误代码
       PrintFormat("Failed to copy data from the iMomentum indicator, error code %d",Ge
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iMFI

函数返回资金流向指标处理器。

```
int  iMFI(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  ma_period,           // 平均周期
   ENUM_APPLIED_VOLUME  applied_volume       // 计算的交易量类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in]计算的平均周期（计算柱）。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 中任意值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Money Flo

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iMFI标图
#property indicator_label1  "iMFI"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 指标窗口的横向水平
#property indicator_level1  20
#property indicator_level2  80
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMFI,              // 使用iMFI
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iMFI;               // 函数类型
input int                   ma_period=14;                 // 周期
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // 交易量类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          iMFIBuffer[];
//--- 存储iMFI指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在资金流向指数指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iMFIBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 创建指标处理程序
   if(type==Call_iMFI)
      handle=iMFI(name,period,ma_period,applied_volume);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[2];
      //--- 周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 交易量类型
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_MFI,2,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iMFI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示资金流向指数指标计算的交易品种/时间帧
   short_name=StringFormat("iMFI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period, EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iMFI 指标复制的值数
   int values_to_copy;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iMFI 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iMFIBuffer 数组大于交易品种/周期 iMFI 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以资金流向指数指标的值填充iMFIBuffer数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iMFIBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住资金流向指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iMFI 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // 资金流向指数值的指标缓冲区
                         int ind_handle,     // iMFI 指标的处理程序
                         int amount          // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分 iMFIBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 如果复制失败，显示错误代码
       PrintFormat("Failed to copy data from the iMFI indicator, error code %d",GetLast
       //--- 退出零结果 - 它表示被认为是不计算的指标
       return(false);
      }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iMA

函数返回移动平均数指标处理器。只有一个缓冲区。

```
int  iMA(
   string                symbol,              // 交易品种名称
   ENUM_TIMEFRAMES       period,              // 周期
   int                   ma_period,           // 平均周期
   int                   ma_shift,            // 平移
   ENUM_MA_METHOD        ma_method,           // 平滑类型
   ENUM_APPLIED_PRICE    applied_price        // 价格或者处理程序类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 计算移动平均数的平均周期。

*ma_shift*

[in] 与价格图表有关的指标变化。

*ma_method*

[in] 平滑类型。可以是 ENUM_MA_METHOD 值中的一个。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                     Demo_iMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Moving Average."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iMA 标图
#property indicator_label1  "iMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMA,                // 使用iMA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iMA;                // 函数类型
input int                   ma_period=10;                 // ma周期
input int                   ma_shift=0;                   // 移动
input ENUM_MA_METHOD        ma_method=MODE_SMA;           // 平滑类型
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;    // 价格类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          iMABuffer[];
//--- 存储iMA指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在移动平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iMABuffer,INDICATOR_DATA);
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iMA)
      handle=iMA(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- 周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- 平滑类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MA,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iMA indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示移动平均指标计算的交易品种/时间帧
   short_name=StringFormat("iMA(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                           ma_period, ma_shift,EnumToString(ma_method),EnumToString(ap
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iMA指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iMA 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iMABuffer数组大于交易品种/周期iMA指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以适合移动平均指标的值填充iMABuffer 数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住移动平均指标的数量值
   bars_calculated=calculated;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充MA指标的指标缓冲区                                           |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // 移动平均值的指标缓冲区
                         int shift,          // 移动
                         int ind_handle,     // iMA指标的处理程序
                         int amount          // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iMABuffer数组
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iMA indicator, error code %d",GetLastE
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iOsMA

函数返回移动平均震荡指标处理器。OsMA振荡器显示MACD与其讯息线之间的区别。只有一个缓冲区。

```
int  iOsMA(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  fast_ema_period,     // 快速移动平均数周期
   int                  slow_ema_period,     // 慢速移动平均数周期
   int                  signal_period,       // 不同点的平均周期
   ENUM_APPLIED_PRICE   applied_price        // 价格或者处理器的类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*fast_ema_period*

[in] 快速移动平均数计算周期。

*slow_ema_period*

[in] 慢速移动平均数计算周期。

*signal_period*

[in] 讯息线计算平均周期。

*applied_price*

[in] 使用价格。可以是任意ENUM_APPLIED_PRICE价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

在一些系统中该振荡器也被认为是MACD柱状图。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iOsMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOsMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Moving Ave

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iOsMA 标图
#property indicator_label1  "iOsMA"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOsMA,              // 使用iOsMA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iOsMA;            // 函数类型
input int                   fast_ema_period=12;         // 快速ma周期
input int                   slow_ema_period=26;         // 慢速ma周期
input int                   signal_period=9;            // 平均差异周期
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iOsMABuffer[];
//--- 存储 iAMA 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在移动平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iOsMABuffer,INDICATOR_DATA);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iOsMA)
      handle=iOsMA(name,period,fast_ema_period,slow_ema_period,signal_period,applied_p
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- 快速ma周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- 慢速ma周期
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- 快速慢速移动平均差异的周期
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_OSMA,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create a handle of iOsMA for the pair %s/%s, error code
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示移动平均震荡指标计算的交易品种/时间帧
   short_name=StringFormat("iOsMA(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                           fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iOsMA 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iOsMA 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iOsMABuffer 数组大于交易品种/周期iOsMA指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iOsMA 指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iOsMABuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 记住移动平均震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iOsMA 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  // OsMA 值的指标缓冲区
                         int ind_handle,        // iOsMA 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iOsMABuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,ama_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iOsMA indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iMACD

函数返回移动平均数聚/散指标处理器。在OsMA被称为MACD柱状图的系统中，该指标显示为两条线。在客户端移动平均数聚/散像柱状图。

```
int  iMACD(
   string               symbol,              // 交易品种名称
   ENUM_TIMEFRAMES      period,              // 周期
   int                  fast_ema_period,     // 快速移动平均数周期
   int                  slow_ema_period,     // 慢速移动平均数周期
   int                  signal_period,       // 不同点的平均周期
   ENUM_APPLIED_PRICE   applied_price        // 价格或者处理器的类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*fast_ema_period*

[in] 快速移动平均数计算周期。

*slow_ema_period*

[in] 慢速移动平均数计算周期。

*signal_period*

[in] 讯号线计算周期。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0 - MAIN_LINE，1 - SIGNAL_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iMACD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMACD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard MACD."

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- MACD 标图
#property indicator_label1  "MACD"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 信号标图
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMACD,              // 使用iMACD
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iMACD;            // 函数类型
input int                   fast_ema_period=12;         // 快速ma周期
input int                   slow_ema_period=26;         // 慢速ma周期
input int                   signal_period=9;            // 平均差异周期
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double         MACDBuffer[];
double         SignalBuffer[];
//--- 存储iMACD 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在移动平均聚/散指标中保持值的数量
int    bars_calculated=0;
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,MACDBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iMACD)
      handle=iMACD(name,period,fast_ema_period,slow_ema_period,signal_period,applied_p
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- 快速ma周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- 慢速ma周期
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- 快速慢速移动平均差异的周期
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MACD,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iMACD indicator for the symbol %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
```

© 2000-2025, MetaQuotes Ltd.

```
      return(INIT_FAILED);
     }
//--- 显示移动平均聚/散指标计算的交易品种/时间帧
   short_name=StringFormat("iMACD(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                           fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从 iMACD 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iMACD 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 MACDBuffer数组大于交易品种/周期iMACD指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iMACD指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!FillArraysFromBuffers(MACDBuffer,SignalBuffer,handle,values_to_copy)) return(0)
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住移动平均聚/散指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iMACD 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &macd_buffer[],     // MACD值的指标缓冲区
                           double &signal_buffer[],   // MACD 信号线的指标缓冲区
                           int ind_handle,            // iMACD 指标的处理程序
                           int amount                 // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iMACDBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,macd_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }

//--- 以1标引指标缓冲区的值填充部分SignalBuffer数组
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
    if(handle!=INVALID_HANDLE)
       IndicatorRelease(handle);
//--- 删除指标后清空图表
    Comment("");
   }
```

© 2000-2025, MetaQuotes Ltd.

# iOBV

函数返回平衡交易量指标处理器。只有一个缓冲区。

```
int  iOBV(
   string                symbol,              // 交易品种名称
   ENUM_TIMEFRAMES       period,              // 周期
   ENUM_APPLIED_VOLUME   applied_volume       // 计算的交易量类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*applied_volume*

[in] 使用的交易量，可以是 ENUM_APPLIED_VOLUME 中任意值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iOBV.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOBV technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---  iOBV
#property indicator_label1  "iOBV"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOBV ,              // 使用iOBV
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iOBV;                // 函数类型
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;    // 交易量类型
input string                symbol=" ";                    // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;         // 时间帧
//--- 指标缓冲区
double          iOBVBuffer[];
//--- 存储iOBV指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在平衡交易量指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iOBVBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iOBV)
      handle=iOBV(name,period,applied_volume);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- 交易量类型
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_OBV,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iOBV indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示平衡交易量指标计算的交易品种/时间帧
   short_name=StringFormat("iOBV(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iOBV 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iOBV 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果iOBVBuffer 数组大于交易品种/周期iOBV 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iOBV 指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iOBVBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住平衡交易量指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iOBV指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &obv_buffer[],  // OBV 值的指标缓冲区
                         int ind_handle,        // iOBV 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iOBVBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,obv_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iOBV indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iSAR

函数返回抛物转向系统指标处理器。只有一个缓冲区。

```
int  iSAR(
   string           symbol,      // 交易品种名称
   ENUM_TIMEFRAMES  period,      // 周期
   double           step,        // 逐步增加
   double           maximum      // 最大止损水平
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*step*

[in] 停止水平增量，通常是0.02。

*maximum*

[in] 最大停止水平，通常是0.2。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iSAR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iSAR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Parabolic

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 绘制iSAR
#property indicator_label1  "iSAR"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iSAR,              // 使用iSAR
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iSAR;              // 函数类型
input double                step=0.02;                   // 步骤 - 停止拖拽的加速因素
input double                maximum=0.2;                 // 步骤的最大值
input string                symbol=" ";                  // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;       // 时间帧
//--- 指标缓冲区
double          iSARBuffer[];
//--- 存储iSAR 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在抛物线转向指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iSARBuffer,INDICATOR_DATA);
//--- 从Wingdings字符集为PLOT_ARROW属性设置图表展示的交易品种代码
   PlotIndexSetInteger(0,PLOT_ARROW,159);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
```

© 2000-2025, MetaQuotes Ltd.

```
   if(type==Call_iSAR)
      handle=iSAR(name,period,step,maximum);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[2];
      //--- 步骤值
      pars[0].type=TYPE_DOUBLE;
      pars[0].double_value=step;
      //--- 可以用于计算的步骤值的限制
      pars[1].type=TYPE_DOUBLE;
      pars[1].double_value=maximum;
      handle=IndicatorCreate(name,period,IND_SAR,2,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iSAR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示抛物线转向指标计算的交易品种/时间帧
   short_name=StringFormat("iSAR(%s/%s, %G, %G)",name,EnumToString(period),
                           step,maximum);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iSAR指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
```

© 2000-2025, MetaQuotes Ltd.

```
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iSAR 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iSARBuffer数组大于交易品种/周期iSAR 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iSAR 指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iSARBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住抛物线转向指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iSAR 指标的指标缓冲区                                          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &sar_buffer[],  // 抛物线转向值的指标缓冲区
                         int ind_handle,        // iSAR 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iSARBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iSAR indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iRSI

函数返回相对强度指数指标处理器。只有一个缓冲区。

```
int  iRSI(
   string               symbol,            // 交易品种类型
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] RSI计算平均周期。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iRSI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRSI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Relative S

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- 绘制iRSI
#property indicator_label1  "iRSI"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 在指标窗口显示值的限制
#property indicator_maximum 100
#property indicator_minimum 0
//--- 指标窗口的横向水平
#property indicator_level1  70.0
#property indicator_level2  30.0
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRSI,              // 使用iRSI
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iRSI;              // 函数类型
input int                   ma_period=14;                // 平均移动周期
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;   // 价格类型
input string                symbol=" ";                  // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;       // 时间帧
//--- 指标缓冲区
double          iRSIBuffer[];
//--- 存储iRSI指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在相对强度指数指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iRSIBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iRSI)
      handle=iRSI(name,period,ma_period,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[2];
      //--- 平均移动周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 可以用于计算的步骤值的限制
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_RSI,2,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iRSI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示相对强度指数指标计算的交易品种/时间帧
   short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),
                           ma_period,applied_price);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
```

© 2000-2025, MetaQuotes Ltd.

```
  {
//--- 从iRSI 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果 iRSI 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iRSIBuffer 数组大于交易品种/周期iRSI指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iRSI指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iRSIBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住相对强度指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iRSI 指标的指标缓冲区                                          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &rsi_buffer[],  // 相对强度指数值的指标缓冲区
                         int ind_handle,        // iRSI 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
```

© 2000-2025, MetaQuotes Ltd.

```
   ResetLastError();
//---以0标引指标缓冲区的值填充部分iRSIBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,rsi_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iRSI indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iRVI

函数返回相对活力指数指标处理器。

```
int  iRVI(
   string           symbol,        // 交易品种名称
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] RVI计算平均周期。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码如下：0 - MAIN_LINE，1 - SIGNAL_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iRVI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRVI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Relative V

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- RVI 标图
#property indicator_label1  "RVI"
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 信号标图
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRVI,              // 使用iRVI
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iRVI;              // 函数类型
input int                   ma_period=10;                // 计算周期
input string                symbol=" ";                  // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;       // 时间帧
//--- 指标缓冲区
double         RVIBuffer[];
double         SignalBuffer[];
//--- 存储iRVI指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在相对活力指数指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,RVIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 获得指标附属的图表交易品种
       name=_Symbol;
      }
//--- 创建指标处理程序
   if(type==Call_iRVI)
      handle=iRVI(name,period,ma_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- 计算周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_RVI,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iRVI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示相对活力指数指标计算的交易品种/时间帧
   short_name=StringFormat("iRVI(%s/%s, %d, %d)",name,EnumToString(period),ma_period);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iRVI 指标复制的值数
   int values_to_copy;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iRVI指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果RVIBuffer数组大于交易品种/周期iRVI 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iRVI 指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(RVIBuffer,SignalBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住相对活力指数指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iRVI 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &rvi_buffer[],     // 相对活力指数值的指标缓冲区
                         double &signal_buffer[],  // 信号线的指标缓冲区
                         int ind_handle,           // iRVI指标的处理程序
                         int amount                // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iRVIBuffer数组
```

© 2000-2025, MetaQuotes Ltd.

```
   if(CopyBuffer(ind_handle,0,0,amount,rvi_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 以1标引指标缓冲区的值填充部分SignalBuffer数组
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iStdDev

函数返回标准偏差指标处理器。只有一个缓冲区。

```
int  iStdDev(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   int                  ma_shift,          // 平移
   ENUM_MA_METHOD       ma_method,         // 平滑类型
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 指标计算平均周期。

*ma_shift*

[in] 与价格图表有关的指标变化。

*ma_method*

[in] 平均类型。可以使 ENUM_MA_METHOD 中任意值。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iStdDev.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStdDev technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the normal Standard Dev

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iStdDev 标图
#property indicator_label1  "iStdDev"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStdDev,            // 使用iStdDev
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iStdDev;          // 函数类型
input int                   ma_period=20;               // 平均周期
input int                   ma_shift=0;                 // 移动
input ENUM_MA_METHOD        ma_method=MODE_SMA;         // 平滑类型
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iStdDevBuffer[];
//--- 存储 iStdDev 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在标准偏差指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iStdDevBuffer,INDICATOR_DATA);
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iStdDev)
      handle=iStdDev(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- 周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- 移动
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- 平滑类型
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_STDDEV,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iStdDev indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示标准偏差指标计算的交易品种/时间帧
   short_name=StringFormat("iStdDev(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(ma_method),EnumToString(app
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iStdDev指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iStdDev指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果 iStdDevBuffer 数组大于交易品种/周期iStdDev 指标的数量值，那么我们不会复制任何
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以标准偏差指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iStdDevBuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住标准偏差指标的数量值
   bars_calculated=calculated;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iStdDev指标的指标缓冲区                                      |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &std_buffer[],  // 标准偏差线的指标缓冲区
                         int std_shift,         // 标准偏差线的移动
                         int ind_handle,        // iStdDev 指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iStdDevBuffer数组
   if(CopyBuffer(ind_handle,0,-std_shift,amount,std_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iStdDev indicator, error code %d",GetL
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iStochastic

函数返回随机摆动指标处理器。

```
int  iStochastic(
   string            symbol,          // 交易品种名称
   ENUM_TIMEFRAMES   period,          // 周期
   int               Kperiod,         // K线周期（用于计算的柱数）
   int               Dperiod,         // D线周期（开始平滑周期）
   int               slowing,         // 最终平滑
   ENUM_MA_METHOD    ma_method,       // 平滑类型
   ENUM_STO_PRICE    price_field      // 随机计算法
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*Kperiod*

[in] %K 线计算平均周期（计算柱）。

*Dperiod*

[in] %D 线计算平均周期（计算柱）。

*slowing*

[in] 减速值。

*ma_method*

[in] 平均类型。可以是 ENUM_MA_METHOD 中任意值。

*price_field*

[in] 计算价格选择参量。可以是 ENUM_STO_PRICE 值中的一个。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

注释

缓冲区代码：0 - MAIN_LINE，1 - SIGNAL_LINE。

例如：

```
//+------------------------------------------------------------------+
//|                                                Demo_iStochastic.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStochastic technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Stochastic

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- 随机标图
#property indicator_label1  "Stochastic"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- 信号标图
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- 设置指标值的限制
#property indicator_minimum 0
#property indicator_maximum 100
//--- 指标窗口的横向水平
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStochastic,       // 使用iStochastic
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iStochastic;     // 函数类型
input int                  Kperiod=5;                 // K 线周期（用于计算的柱数）
input int                  Dperiod=3;                 // D 线周期(主要平滑周期)
input int                  slowing=3;                 // 最后平滑周期
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // 平滑类型
input ENUM_STO_PRICE       price_field=STO_LOWHIGH;   // 随机计算方法
input string               symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
```

© 2000-2025, MetaQuotes Ltd.

```
double         StochasticBuffer[];
double         SignalBuffer[];
//--- 存储iStochastic指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在随机震荡指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,StochasticBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iStochastic)
      handle=iStochastic(name,period,Kperiod,Dperiod,slowing,ma_method,price_field);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[5];
      //--- 用于计算的K线周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=Kperiod;
      //--- 用于主要平滑的D线周期
      pars[1].type=TYPE_INT;
      pars[1].integer_value=Dperiod;
      //--- 最后平滑的K线周期
      pars[2].type=TYPE_INT;
      pars[2].integer_value=slowing;
      //--- 平滑类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ma_method;
      //--- 随机计算方法
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[4].type=TYPE_INT;
      pars[4].integer_value=price_field;
      handle=IndicatorCreate(name,period,IND_STOCHASTIC,5,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iStochastic indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示随机震荡指标计算的交易品种/时间帧
   short_name=StringFormat("iStochastic(%s/%s, %d, %d, %d, %s, %s)",name,EnumToString
                           Kperiod,Dperiod,slowing,EnumToString(ma_method),EnumToStrin
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iStochastic指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iStochastic指标数量值更改
//---或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 如果StochasticBuffer数组大于交易品种/周期iStochastic 指标的数量值，那么我们不会复制
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iStochastic 指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(StochasticBuffer,SignalBuffer,handle,values_to_copy)) ret
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住随机震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iStochastic指标的指标缓冲区                                  |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &main_buffer[],    // 随机震荡值的指标缓冲区
                           double &signal_buffer[],  // 信号线的指标缓冲区
                           int ind_handle,           // iStochastic 指标的处理程序
                           int amount                // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分StochasticBuffer数组
   if(CopyBuffer(ind_handle,MAIN_LINE,0,amount,main_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 以1标引指标缓冲区的值填充部分SignalBuffer数组
   if(CopyBuffer(ind_handle,SIGNAL_LINE,0,amount,signal_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iTEMA

函数返回三倍指数移动平均指标处理器。只有一个缓冲区。

```
int  iTEMA(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   int                  ma_shift,          // 指标平移
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*ma_period*

[in] 计算平均周期（计算柱）。

*ma_shift*

[in] 与价格图表相关的指标变化。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iTEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Triple Exp

#property indicator_chart_window
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_buffers 1
#property indicator_plots   1
//--- iTEMA 标图
#property indicator_label1  "iTEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTEMA,              // 使用iTEMA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iTEMA;            // 函数类型
input int                   ma_period=14;               // 平均周期
input int                   ma_shift=0;                 // 移动
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;  // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iTEMABuffer[];
//--- 存储iTEMA 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在三倍指数移动平均指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iTEMABuffer,INDICATOR_DATA);
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 获得指标附属的图表交易品种
       name=_Symbol;
      }
//--- 创建指标处理程序
   if(type==Call_iTEMA)
      handle=iTEMA(name,period,ma_period,ma_shift,applied_price);
   else
      {
       //--- 以指标参数填充结构
       MqlParam pars[3];
       //--- 周期
       pars[0].type=TYPE_INT;
       pars[0].integer_value=ma_period;
       //--- 移动
       pars[1].type=TYPE_INT;
       pars[1].integer_value=ma_shift;
       //--- 价格类型
       pars[2].type=TYPE_INT;
       pars[2].integer_value=applied_price;
       handle=IndicatorCreate(name,period,IND_TEMA,3,pars);
      }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
      {
       //--- 叙述失败和输出错误代码
       PrintFormat("Failed to create handle of the iTEMA indicator for the symbol %s/%s
                   name,
                   EnumToString(period),
                   GetLastError());
       //--- 指标提前停止
       return(INIT_FAILED);
      }
//--- 显示三倍指数移动平均指标计算的交易品种/时间帧
   short_name=StringFormat("iTEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
```

© 2000-2025, MetaQuotes Ltd.

```
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iTEMA 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iTEMA 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iTEMABuffer数组大于交易品种/周期iTEMA 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以三倍指数移动平均指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iTEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住三倍指数移动平均指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iTEMA 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &tema_buffer[], // 三倍指数移动平均值的指标缓冲区
                         int t_shift,           // 线的移动
```

© 2000-2025, MetaQuotes Ltd.

```
                           int ind_handle,         // iTEMA 指标的处理程序
                           int amount              // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iTEMABuffer数组
   if(CopyBuffer(ind_handle,0,-t_shift,amount,tema_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iTEMA indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iTriX

函数返回三倍指数移动平均数振荡指标处理器。只有一个缓冲区。

```
int  iTriX(
   string               symbol,            // 交易品种名称
   ENUM_TIMEFRAMES      period,            // 周期
   int                  ma_period,         // 平均周期
   ENUM_APPLIED_PRICE   applied_price      // 价格或者处理器类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL值代表当前交易品种。

*period*

[in]周期值可以是ENUM_TIMEFRAMES值中的一个，0代表当前时间表。

*ma_period*

[in] 计算平均周期（计算柱）。

*applied_price*

[in] 使用价格。可以是任意ENUM_APPLIED_PRICE价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease()。

例如：

```
//+------------------------------------------------------------------+
//|                                                  Demo_iTriX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTriX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iTriX标图
#property indicator_label1  "iTriX"
#property indicator_type1   DRAW_LINE
```

© 2000-2025, MetaQuotes Ltd.

```
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTriX,              // 使用iTriX
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iTriX;            // 函数类型
input int                   ma_period=14;               // 周期
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                 // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // 时间帧
//--- 指标缓冲区
double          iTriXBuffer[];
//--- 存储iTriX 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在三倍指数移动平均震荡指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iTriXBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iTriX)
      handle=iTriX(name,period,ma_period,applied_price);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 以指标参数填充结构
       MqlParam pars[2];
       //--- 周期
       pars[0].type=TYPE_INT;
       pars[0].integer_value=ma_period;
       //--- 价格类型
       pars[1].type=TYPE_INT;
       pars[1].integer_value=applied_price;
       handle=IndicatorCreate(name,period,IND_TRIX,2,pars);
      }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
      {
       //--- 叙述失败和输出错误代码
       PrintFormat("Failed to create handle of the iTriX indicator for the symbol %s/%s
                   name,
                   EnumToString(period),
                   GetLastError());
       //--- 指标提前停止
       return(INIT_FAILED);
      }
//--- 显示三倍指数移动平均震荡指标计算的交易品种/时间帧
   short_name=StringFormat("iTriX(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iTriX指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
      {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iTriX指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iTriXBuffer 数组大于交易品种/周期 iTriX 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以三倍指数移动平均震荡指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iTriXBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住三倍指数移动平均震荡指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iTriX 指标的指标缓冲区                                       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &trix_buffer[], // 三倍指数移动平均震荡值的指标缓冲区
                         int ind_handle,        // iTriX指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iTriXBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,trix_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iTriX indicator, error code %d",GetLas
      //--- 退出零结果 - 它表示被认为是不计算的指标
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iWPR

函数返回威廉指数指标。只有一个缓冲区。

```
int  iWPR(
   string           symbol,          // 交易品种名称
   ENUM_TIMEFRAMES  period,          // 周期
   int              calc_period      // 平均周期
   );
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*calc_period*

[in] 指标计算周期（计算柱）。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                   Demo_iWPR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iWPR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iWPR 标图
#property indicator_label1  "iWPR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrCyan
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置指标值的限制
#property indicator_minimum -100
#property indicator_maximum 0
//--- 指标窗口的横向水平
#property indicator_level1  -20.0
#property indicator_level2  -80.0
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iWPR,              // 使用iWPR
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation             type=Call_iWPR;           // 函数类型
input int                  calc_period=14;           // 周期
input string               symbol=" ";               // 交易品种
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;    // 时间帧
//--- 指标缓冲区
double         iWPRBuffer[];
//--- 存储iWPR 指标处理程序的变量
int    handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在Larry Williams百分比范围指标中保持值的数量
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iWPRBuffer,INDICATOR_DATA);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iWPR)
```

© 2000-2025, MetaQuotes Ltd.

```
      handle=iWPR(name,period,calc_period);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- 周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=calc_period;
      handle=IndicatorCreate(name,period,IND_WPR,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iWPR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示Larry Williams百分比范围指标计算的交易品种/时间帧
   short_name=StringFormat("iWPR(%s/%s, %d)",name,EnumToString(period),calc_period);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                               |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iWPR指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 如果它是指标计算的最初起点或如果iWPR 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iWPRBuffer 数组大于交易品种/周期iWPR 指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以威廉姆斯百分比范围指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iWPRBuffer,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住Larry Williams百分比范围指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iWPR指标的指标缓冲区                                         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &wpr_buffer[],  // 威廉姆斯百分比范围值的指标缓冲区
                         int ind_handle,        // iWPR指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//---以0标引指标缓冲区的值填充部分iWPRBuffer数组
   if(CopyBuffer(ind_handle,0,0,amount,wpr_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iWPR indicator, error code %d",GetLast
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iVIDyA

函数返回变量指数动态平均数指标处理器。只有一个缓冲区。

```
int  iVIDyA(
   string               symbol,           // 交易品种名称
   ENUM_TIMEFRAMES      period,           // 周期
   int                  cmo_period,       // Chande 动量指标周期
   int                  ema_period,       // EMA 平滑周期
   int                  ma_shift,         // 价格图表平移
   ENUM_APPLIED_PRICE   applied_price     // 价格或者处理器类型
   );
```

参量

*symbol*

[in]证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

*period*

[in]周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*cmo_period*

[in] 钱德动量摆动指标计算周期（计算柱）。

*ema_period*

[in] 平滑系数计算EMA周期（计算柱）。

*ma_shift*

[in] 与价格图表相关的指标变化。

*applied_price*

[in] 使用价格。可以是任意 ENUM_APPLIED_PRICE 价格常量或者另外指标处理器。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                 Demo_iVIDyA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVIDyA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Variable Index Dynami

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iVIDyA 标图
#property indicator_label1  "iVIDyA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iVIDyA,             // 使用iVIDyA
   Call_IndicatorCreate     // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iVIDyA;          // 函数类型
input int                   cmo_period=15;             // Chande 动量周期
input int                   ema_period=12;             // 平滑因素周期
input int                   ma_shift=0;                // 移动
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // 价格类型
input string                symbol=" ";                // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // 时间帧
//--- 指标缓冲区
double          iVIDyABuffer[];
//--- 存储iVIDyA 指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在变量指数动态平均指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iVIDyABuffer,INDICATOR_DATA);
//--- 设置移动
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- 定义绘制指标的交易品种
   name=symbol;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iVIDyA)
      handle=iVIDyA(name,period,cmo_period,ema_period,ma_shift,applied_price);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[4];
      //--- Chande 动量周期
      pars[0].type=TYPE_INT;
      pars[0].integer_value=cmo_period;
      //--- 平滑因素周期
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ema_period;
      //--- 移动
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_shift;
      //--- 价格类型
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_VIDYA,4,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iVIDyA indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示变量指数动态平均指标计算的交易品种/时间帧
   short_name=StringFormat("iVIDyA(%s/%s, %d, %d, %d, %s)",name,EnumToString(period),
                           cmo_period,ema_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| 自定义指标迭代函数                                                |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iVIDyA 指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iVIDyA 指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iVIDyABuffer数组大于交易品种/周期iVIDyA指标的数量值，那么我们不会复制任何内容
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()）最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以变量指数动态平均指标的值填充数组
//--- 如果FillArrayFromBuffer 返回 false，它表示信息还未准备，退出操作
   if(!FillArrayFromBuffer(iVIDyABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住变量指数动态平均指标的数量值
   bars_calculated=calculated;
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充iVIDyA 指标的指标缓冲区                                      |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &vidya_buffer[],// 变量指数动态平均值的指标缓冲区
                         int v_shift,           // 线的移动
                         int ind_handle,        // iVIDyA指标的处理程序
                         int amount             // 复制值的数量
                         )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iVIDyABuffer 数组
   if(CopyBuffer(ind_handle,0,-v_shift,amount,vidya_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iVIDyA indicator, error code %d",GetLa
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# iVolumes

函数返回交易量指标处理器。只有一个缓冲区。

```
int  iVolumes(
   string                symbol,              // 交易品种名称
   ENUM_TIMEFRAMES       period,              // 周期
   ENUM_APPLIED_VOLUME   applied_volume       // 计算的交易量类型
   )
```

参量

*symbol*

[in] 证券交易品种名称，数据用来计算指标。NULL 值代表当前交易品种。

[in] 周期值可以是 ENUM_TIMEFRAMES 值中的一个，0代表当前时间表。

*applied_volume*

[in] 使用的交易量。可以是 ENUM_APPLIED_VOLUME 的任意值。

返回值

返回特殊技术指标处理器，失败返回 INVALID_HANDLE. 计算机内存从不使用的指标中释放，使用指标处理程序传递到的函数 IndicatorRelease() 。

例如：

```
//+------------------------------------------------------------------+
//|                                                 Demo_iVolumes.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVolumes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- iVolumes标图
#property indicator_label1  "iVolumes"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| 枚举处理创建方法                                                 |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iVolumes,          // 使用iVolumes
   Call_IndicatorCreate    // 使用IndicatorCreate
  };
//--- 输入参数
input Creation              type=Call_iVolumes;           // 函数类型
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // 交易量类型
input string                symbol=" ";                   // 交易品种
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // 时间帧
//--- 指标缓冲区
double          iVolumesBuffer[];
double          iVolumesColors[];
//--- 存储iVolumes指标处理程序的变量
int     handle;
//--- 存储变量
string name=symbol;
//--- 图表上的指标名称
string short_name;
//--- 我们将在交易量指标中保持值的数量
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| 自定义指标初始化函数                                             |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 分配指标缓冲区数组
   SetIndexBuffer(0,iVolumesBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iVolumesColors,INDICATOR_COLOR_INDEX);
//--- 定义绘制指标的交易品种
   name=symbol;
//--- 删除向左和向右的空格
   StringTrimRight(name);
   StringTrimLeft(name);
//--- 如果它返回 'name' 字符串的零长度
   if(StringLen(name)==0)
     {
      //--- 获得指标附属的图表交易品种
      name=_Symbol;
     }
//--- 创建指标处理程序
   if(type==Call_iVolumes)
      handle=iVolumes(name,period,applied_volume);
   else
     {
      //--- 以指标参数填充结构
      MqlParam pars[1];
      //--- 价格类型
```

© 2000-2025, MetaQuotes Ltd.

```
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_VOLUMES,1,pars);
     }
//--- 如果没有创建处理程序
   if(handle==INVALID_HANDLE)
     {
      //--- 叙述失败和输出错误代码
      PrintFormat("Failed to create handle of the iVolumes indicator for the symbol %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- 指标提前停止
      return(INIT_FAILED);
     }
//--- 显示交易量指标计算的交易品种/时间帧
   short_name=StringFormat("iVolumes(%s/%s, %s)",name,EnumToString(period),EnumToStrin
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- 指标正常初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 自定义指标迭代函数                                                 |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- 从iVolumes指标复制的值数
   int values_to_copy;
//--- 确定指标计算的数量值
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- 如果它是指标计算的最初起点或如果iVolumes指标数量值更改
//--- 或如果需要计算两个或多个柱形的指标 这意味着价格历史中有些内容会发生变化）
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- 如果iVolumesBuffer数组大于交易品种/周期iVolumes指标的数量值，那么我们不会复制任何内
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 否则，我们复制小于指标缓冲区的大小
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- 它意味着这不是初次指标计算，因为 OnCalculate()) 最近调用
      //--- 为了计算，添加不超过一柱
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- 以iVolumes指标的值填充数组
//--- 如果FillArraysFromBuffer返回false，它表示信息还未准备，退出操作
   if(!FillArraysFromBuffers(iVolumesBuffer,iVolumesColors,handle,values_to_copy)) ret
//--- 形成信息
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- 在图表上展示服务信息
   Comment(comm);
//--- 记住交易量指标的数量值
   bars_calculated=calculated;
//--- 返回prev_calculated值以便下次调用
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| 填充 iVolumes 指标的指标缓冲区                                   |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &volume_buffer[],   // 交易量值的指标缓冲区
                           double &color_buffer[],    // 颜色指标缓冲区
                           int ind_handle,            // iVolumes 指标的处理程序
                           int amount                 // 复制值的数量
                           )
  {
//--- 重置错误代码
   ResetLastError();
//--- 以0标引指标缓冲区的值填充部分iVolumesBuffer 数组
   if(CopyBuffer(ind_handle,0,0,amount,volume_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",Get
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 以1标引指标缓冲区的值填充部分iVolumesColors 数组
   if(CopyBuffer(ind_handle,1,0,amount,color_buffer)<0)
     {
      //--- 如果复制失败，显示错误代码
      PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",Get
```

© 2000-2025, MetaQuotes Ltd.

```
      //--- 退出零结果 - 它表示被认为是不计算的指标
      return(false);
     }
//--- 一切顺利
   return(true);
  }
//+------------------------------------------------------------------+
//| 指标去初始化函数                                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- 删除指标后清空图表
   Comment("");
  }
```

© 2000-2025, MetaQuotes Ltd.

# 处理优化结果

组织定制处理策略测试中优化结果的函数。他们可以在优化期间，在测试代理中调用以及EA交易和脚本中本地调用。

当您在策略测试中运行EA交易时，您可以基于简单类型或简单结构创建您自己的数据数组(它们不包含字符串，类对象或动态数组的对象)。该数据设置可以在称为帧的指定结构中的FrameAdd()函数进行保存。EA交易优化期间，每个代理发送一系列帧到程序端。接收的所有帧都写在terminal_directory/MQL5/Files/Tester expert文件夹的*.MQD文件中，名为Expert Advisor。它们按照从代理接收的顺序编写。在客户端从测试代理接收的帧生成TesterPass事件。

这些帧可以存储在计算机内存和指定名称的文件中。MQL5语言对帧数无设限。

## MQL5云网络中的内存和磁盘空间限制

以下限制适用于在MQL5云网络中运行的优化：EA交易不得向磁盘写入超过1GB的信息或使用超过4GB的内存(RAM)。如果超出限制，网络代理将无法正确完成计算，您将不会收到结果。但是，您需要为计算所花费的所有时间付费。

如果您需要从每个优化过程获取信息，请发送帧而不写入磁盘。为了避免在MQL5云网络计算期间在EA交易中使用文件操作，您可以使用以下检查：

```
int handle=INVALID_HANDLE;
bool file_operations_allowed=true;
if(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_FORWARD))
   file_operations_allowed=false;

if(file_operations_allowed)
  {
   ...
   handle=FileOpen(...);
   ...
  }
```

| 函数 | 动作 |
| --- | --- |
| FrameFirst | 移动阅读帧的指针至起点并重置之前设置的过滤器 |
| FrameFilter | 设置帧阅读过滤器并移动指针至起点 |
| FrameNext | 阅读帧并移动指针至下一个 |
| FrameInputs | 接收形成帧的 输入参数 |
| FrameAdd | 添加数据帧 |
| ParameterGetRange | 在策略测试优化EA期间，接收范围值的数据和输入变量的改变步骤 |
| ParameterSetRange | 在策略测试优化EA期间，指定使用输入变量 ：值，改变步骤，初始值和最终值 |

© 2000-2025, MetaQuotes Ltd.

另见

测试统计，运行MQL5程序的属性

© 2000-2025, MetaQuotes Ltd.

# FrameFirst

移动阅读帧的指针至起点并重置之前设置的过滤器。

```
bool  FrameFirst();
```

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用 GetLastError() 函数。

© 2000-2025, MetaQuotes Ltd.

# FrameFilter

设置帧阅读过滤器并移动指针至起点。

```
bool  FrameFilter(
   const string  name,          // 公共名称/标签
   long          id             // 公共 ID
   );
```

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError() 函数。

注意

如果空字符串作为第一参数被传递，那么过滤器将只处理数值参数，例如只能查看指定id的帧。如果第二参数的值是ULONG_MAX，那么只有文本过滤器工作。

调用 FrameFilter("", ULONG_MAX) 相当于调用 FrameFirst()，例如，相当于不使用任何过滤器。

© 2000-2025, MetaQuotes Ltd.

# FrameNext

阅读帧并移动指针至下一个。有两个函数变量。

1. 调用接收一个数值

```
bool  FrameNext(
   ulong&   pass,       // 添加帧时的优化传递数
   string&  name,       // 公共名称/标签
   long&    id,         // 公共 ID
   double&  value       // 值
   );
```

2. 调用接收真的所有数据

```
bool  FrameNext(
   ulong&   pass,       // 添加帧时的优化传递数
   string&  name,       // 公共名称/标签
   long&    id,         // 公共 ID
   double&  value,      // 值
   void&    data[]      // 任何类型的数组
   );
```

参数

*pass*

[out] 策略测试中优化期间的传递数。

*name*

[out] 标识符名称。

*id*

[out] 标识符的值。

*value*

[out] 单一数值。

*data*

[out] 任何类型的数组。

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError()函数。

注意

在第二版调用，您必须正确处理*data[]*数组中接收的数据。

© 2000-2025, MetaQuotes Ltd.

# FrameInputs

接收形成指定传递数帧的输入参数。

```
bool  FrameInputs(
   ulong    pass,                // 优化传递数
   string&  parameters[],        // 形成"parameterN=valueN"字符串的数组
   uint&    parameters_count     // 参数总数
   );
```

参数

*pass*

[in] 策略测试中优化期间的传递数。

*parameters*

[out] 描述名称和参数值的字符串数组。

*parameters_count*

[out] 数组*parameters[]*的元素数量。

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError()函数。

注意

获得*parameters[]*数组中的*parameters_count*字符串数以后，您可以通过所有记录组织一个循环。这将帮您找到指定传递数的EA输入参数的值。

© 2000-2025, MetaQuotes Ltd.

# FrameAdd

添加数据帧。有两个函数变量。

1. 添加文件数据

```
bool  FrameAdd(
   const string  name,        // 公共名称/标签
   long          id,          // 公共 ID
   double        value,       // 值
   const string  filename     // 数据文件的名称
   );
```

2. 添加任何类型数组的数据

```
bool  FrameAdd(
   const string  name,        // 公共名称/标签
   long          id,          // 公共 ID
   double        value,       // 值
   const void&   data[]       // 任何类型的数组
   );
```

参数

*name*

[in] 公共帧标签。它可以被用于FrameFilter()函数的过滤器。

*id*

[in] 帧的公共标识符。它可以被用于FrameFilter()函数的过滤器。

*value*

[in] 写入帧的数值。它被用来发送像OnTester()函数的单一传递结果。

*filename*

[in] 包含添加到帧的数据的文件名称。文件必须位于MQL5/Files文件夹。

*data*

[in] 写入帧的任何类型的数组。通过引用传递。

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError()函数。

© 2000-2025, MetaQuotes Ltd.

# ParameterGetRange

在策略测试优化EA期间，接收范围值的数据和输入变量的改变步骤。有两个函数变量。

1. 接收整型输入参数的数据

```
bool  ParameterGetRange(
   const string  name,           // 参数 (输入变量) 名称
   bool&         enable,         // 启用参数优化
   long&         value,          // 参数值
   long&         start,          // 初始值
   long&         step,           // 改变步骤
   long&         stop            // 最终值
   );
```

2. 接收真实类型输入参数的数据

```
bool  ParameterGetRange(
   const string  name,           // 参数(输入变量)名称
   bool&         enable,         // 启用参数优化
   double&       value,          // 参数值
   double&       start,          // 初始值
   double&       step,           // 改变步骤
   double&       stop            // 最终值
   );
```

参数

*name*

[in] 输入变量 ID。这些变量是应用程序的外部参数。它们的值可以在启动图表时或单一测试时指定。

*enable*

[out] 策略测试优化期间，该参数可以用于枚举值的标识。

*value*

[out] 参数值。

*start*

[out] 优化期间的初始参数值。

*step*

[out] 枚举值时的参数变换步骤。

*stop*

[out] 优化期间的最终参数值。

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError()函数。

注意

© 2000-2025, MetaQuotes Ltd.

函数仅能从 OnTesterInit()、OnTesterPass() 和 OnTesterDeinit() 处理程序调用。它已被引入接收EA输入参数值和策略测试优化期间的变量范围。

当调用OnTesterInit()时，获得的数据可以通过ParameterSetRange()函数被用于重新定义任何输入变量的枚举规则。因此，可以设置新的Start，Stop 和 Step值并且无论策略测试如何设置，输入参数甚至可以完全脱离优化。这可以使您根据EA交易的关键参数值，通过排除一些优化参数，管理优化输入参数的区域。

例如：

```
#property description "Expert Advisor for ParameterGetRange() function demonstration."
#property description "Should be launched in the optimization mode of the Strategy Tes
//--- 输入参数
input int                  Input1=1;
input double               Input2=2.0;
input bool                 Input3=false;
input ENUM_DAY_OF_WEEK     Input4=SUNDAY;

//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- EA是只为策略测试操作而设计的
   if(!MQL5InfoInteger(MQL5_OPTIMIZATION))
     {
      MessageBox("Should be launched in the optimization mode of the Strategy Tester!"
      //--- 提前完成EA操作并从图表删除
      return(INIT_FAILED);
     }
//--- 成功完成初始化
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| TesterInit 函数                                                  |
//+------------------------------------------------------------------+
void OnTesterInit()
  {
//--- 长整型输入参数的示例
   string name="Input1";
   bool enable;
   long par1,par1_start,par1_step,par1_stop;
   ParameterGetRange(name,enable,par1,par1_start,par1_step,par1_stop);
   Print("First parameter");
   PrintFormat("%s=%d  enable=%s  from %d to %d with step=%d",
               name,par1,(string)enable,par1_start,par1_stop,par1_step);
//--- 双精度型输入参数的示例
   name="Input2";
   double par2,par2_start,par2_step,par2_stop;
   ParameterGetRange(name,enable,par2,par2_start,par2_step,par2_stop);
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("Second parameter");
   PrintFormat("%s=%G  enable=%s  from %G to %G with step=%G",
               name,par2,(string)enable,par2_start,par2_stop,par2_step);
//--- 布尔型输入参数的示例
   name="Input3";
   long par3,par3_start,par3_step,par3_stop;
   ParameterGetRange(name,enable,par3,par3_start,par3_step,par3_stop);
   Print("Third parameter");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,(string)par3,(string)enable,
               (string)par3_start,(string)par3_stop);
//--- 枚举型输入参数的示例
   name="Input4";
   long par4,par4_start,par4_step,par4_stop;
   ParameterGetRange(name,enable,par4,par4_start,par4_step,par4_stop);
   Print("Fourth parameter");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,EnumToString((ENUM_DAY_OF_WEEK)par4),(string)enable,
               EnumToString((ENUM_DAY_OF_WEEK)par4_start),
               EnumToString((ENUM_DAY_OF_WEEK)par4_stop));
  }
//+------------------------------------------------------------------+
//| TesterDeinit函数                                                 |
//+------------------------------------------------------------------+
void OnTesterDeinit()
  {
//--- 优化完成后将会显示该信息
   Print(__FUNCTION__," Optimization completed");
  }
```

© 2000-2025, MetaQuotes Ltd.

# ParameterSetRange

在策略测试优化EA期间，指定使用输入变量：值，改变步骤，初始值和最终值。有两个函数变量。

1. 指定整型输入参数的数据

```
bool  ParameterSetRange(
   const string  name,           // 参数（输入变量）名称
   bool          enable,         // 启用参数优化
   long          value,          // 参数值
   long          start,          // 初始值
   long          step,           // 改变步骤
   long          stop            // 最终值
   );
```

2. 指定真实类型输入参数的数据

```
bool  ParameterSetRange(
   const string  name,           // 参数(输入变量)名称
   bool          enable,         // 启用参数优化
   double        value,          // 参数值
   double        start,          // 初始值
   double        step,           // 改变步骤
   double        stop            // 最终值
   );
```

参数

*name*

[in] 输入或sinput 变量 ID。这些变量是应用程序的外部参数。它们的值可以在启动程序时指定。

*enable*

[in] 在策略测试优化期间启用该参数以枚举值。

*value*

[in] 参数值。

*start*

[in] 优化期间的初始参数值。

*step*

[in] 枚举值时的参数变换步骤。

*stop*

[in] 优化期间的最终参数值。

返回值

如果成功返回 true，否则 false。若要获得有关错误的信息，请调用GetLastError() 函数。

注意

© 2000-2025, MetaQuotes Ltd.

从策略测试启动优化时，函数仅能从 OnTesterInit() 处理程序调用。它专为指定参数范围和变换步骤而设计的。参数可以从优化完全脱离，而无论策略测试如何设置。它也允许使用优化进程中sinput标识符声明的变量。

根据关键参数值，ParameterSetRange() 函数允许您管理策略测试的EA优化，包括或不包括所需的优化输入参数和设置所需的范围和变换步骤。

© 2000-2025, MetaQuotes Ltd.

# 事件函数

该组包括自定义事件函数和定时器事件函数。除了该组函数外，有处理预定义事件特别函数。

| 函数 | 功能 |
| --- | --- |
| EventSetMillisecondTimer | 为当前图表启动周期少于1秒的高分辨率计时器的事件生成器 |
| EventSetTimer | 用当前图表指定周期启动定时器事件生成器 |
| EventKillTimer | 依据当前图表定时器停止事件生成器 |
| EventChartCustom | 为指定图表生成自定义事件 |

另见

图表事件类型

© 2000-2025, MetaQuotes Ltd.

# EventSetMillisecondTimer

函数显示到客户端，为这个EA或指标的timer事件应当在间隔不到一秒钟就被触发.

```
bool  EventSetMillisecondTimer(
   int  milliseconds      // 毫秒数
   );
```

参数

*milliseconds*

[in] 定义定时器事件的频率的毫秒数.

返回值

若成功返回true，否则false。若要获得错误代码，调用函数GetLastError()。

注意

此功能是为需要高分辨的计时器的情况而设计的。换句话说，定时器事件应当比每秒一次更频繁接收。如果一个超过一秒钟时间的常规计时器对您已经足够，使用 EventSetTimer().

一般来说，当定时器时长缩短，测试时间增加，作为定时器事件处理被称作多于平时。在实时模式下工作时，由于硬件限制，计时器事件在10-16毫秒内运转不超过一次。

一般情况下，这个函数必须从函数OnInit()或者从类构造函数中调用。若为处理定时器事件，EA交易必须有OnTimer()函数。

每个EA交易及标识符使用自己的定时器工作，且仅从那里接收事件。一旦mql5程序停止运行，如果定时器不能通过EventKillTimer()函数创建和禁止，则会被强制销毁。

对于每一个程序只能运行一个定时器。每个mql5程序以及每个图表都有其自己的事件队列，其中放置了所有最近收到的事件。如果 Timer 事件正在队列或者正被处理中，新的Timer事件将不会被放置在mql5程序队列中.

© 2000-2025, MetaQuotes Ltd.

# EventSetTimer

该函数应用于客户端，定时器事件必须依据EA交易依附的图表指定周期生成。

```
bool  EventSetTimer(
   int  seconds      // 秒数
   );
```

参量

*seconds*

[in] 决定定时器事件发生频率的秒数。

返回值

若成功返回true，否则false。若要获得错误代码，调用函数GetLastError()。

注释

一般情况下，这个函数必须从函数OnInit()或者从类构造函数中调用。若为处理定时器事件，EA交易必须有OnTimer()函数。

每个EA交易及标识符使用自己的定时器工作，且仅从那里接收事件。一旦mql5程序停止运行，如果定时器不能通过EventKillTimer()函数创建和禁止，则会被强制销毁。

对于每一个程序只能运行一个定时器。每个mql5程序以及每个图表都有其自己的事件队列，其中放置了所有最近收到的事件。如果 Timer 事件正在队列或者正被处理中，新的Timer事件将不会被放置在mql5程序队列中.

© 2000-2025, MetaQuotes Ltd.

# EventKillTimer

指定停止生成定时器事件的客户端。

```
void  EventKillTimer();
```

返回值

无返回值

注释

特别是，如果函数OnInit()从函数EventSetTimer()调用，该函数必须从函数OnDeinit()调用。如果函数EventSetTimer()在类构造函数中调用，那么该函数也可以从类析构函数调用。

每个EA交易及标识符使用自己的定时器工作，且仅从那里接收事件。一旦mql5程序停止运行，如果定时器不能通过EventKillTimer()函数创建和禁止，则会被强制销毁。

© 2000-2025, MetaQuotes Ltd.

# EventChartCustom

该函数为指定图表生成自定义事件。

```
bool  EventChartCustom(
   long    chart_id,             // 接收图表事件的标识符
   ushort  custom_event_id,      // 事件标识符
   long    lparam,               // 长整型参量
   double  dparam,               // 双精度型参量
   string  sparam                // 事件的字符串参量
   );
```

参量

*chart_id*

[in] 图表标识符。0 意味着当前图表。

*custom_event_id*

[in] 用户事件ID。这个标识符自动添加到CHARTEVENT_CUSTOM值中且转化成整数型。

*lparam*

[in] 传递到OnChartEvent函数的长整型事件参量。

*dparam*

[in] 传递到OnChartEvent函数的双精度型事件参量。

*sparam*

[in] 传递到OnChartEvent函数的字符串型事件参量。如果字符串超过63个字符，则会被删节。

返回值

如果一个自定义事件已经成功被放置在接收到事件的图表队列事件中，返回 true. 若成功返回true，否则false。调用函数GetLastError() 获得错误代码。

注释

附在指定图表的EA交易或者指标处理使用OnChartEvent函数 (int event_id, long& lparam, double& dparam, string& sparam )的事件。

对于每一种事件类型，OnChartEvent()函数的输入参量有事件需要过程的定值。通过这个参量传递的事件和值列在以下表格中。

© 2000-2025, MetaQuotes Ltd.

| 事件 | id参量值 | lparam 参量值 | dparam参量值 | sparam参量值 |
| --- | --- | --- | --- | --- |
| 击键事件 | CHARTEVENT_KEYDOWN | 击键代码 | 重复计数（用户按住按键的重复击键次数） | 描述键盘按键状态的位掩码的字符串值 |
| 鼠标事件 (if property CHART_EVENT_MOUSE_MOVE=true is set for the chart) | CHARTEVENT_MOUSE_MOVE | X 坐标 | Y 坐标 | 描述鼠标按键状态的位掩码的字符串值 |
| 图解物件创建事件 (if CHART_EVENT_OBJECT | CHARTEVENT_OBJECT_CREATE | — | — | 创建的 |

© 2000-2025, MetaQuotes Ltd.

| 事件 | id参量值 | lparam 参量值 | dparam参量值 | sparam参量值 |
|---|---|---|---|---|
| _CREATE=true is set for the chart) | | | | 图解物件名称 |
| 通过属性对话框改变物件属性事件 | CHARTEVENT_OBJECT_CHANGE | — | — | 更改的图解物件名称 |
| 图解物件删除事件 (if CHART_EVENT_OBJECT_DELETE=true is set for the chart) | CHARTEVENT_OBJECT_DELETE | — | — | 删除的图解物件名称 |
| 图表上鼠标点击事件 | CHARTEVENT_CLICK | X 坐标 | Y 坐标 | — |
| 附属于图表的图解物件鼠标点击事件 | CHARTEVENT_OBJECT_CLICK | X 坐标 | Y 坐标 | 事件发生的图解物件 |

© 2000-2025, MetaQuotes Ltd.

| 事件 | id参量值 | lparam 参量值 | dparam参量值 | sparam参量值 |
|---|---|---|---|---|
| | | | | 名称 |
| 用鼠标拖拽图解物件事件 | CHARTEVENT_OBJECT_DRAG | — | — | 移动的图解物件名称 |
| LabelEdit图解物件输入框中完成文本编辑事件 | CHARTEVENT_OBJECT_ENDEDIT | — | — | 完成文本编辑的LabelEdit图解物件名称 |
| 图表中的更改事件 | CHARTEVENT_CHART_CHANGE | — | — | — |
| N号码下用户ID事件 | CHARTEVENT_CUSTOM+N | 通过EventChartCustom()函数设置的值 | 通过EventC | 通过EventC |

© 2000-2025, MetaQuotes Ltd.

| 事件 | id参量值 | lparam 参量值 | dparam参量值 | sparam参量值 |
| --- | --- | --- | --- | --- |
| | | | hartCustom()函数设置的值 | hartCustom()函数设置的值 |

示例：

```
//+------------------------------------------------------------------+
//|                                                ButtonClickExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;
//+------------------------------------------------------------------+
//| 专家初始化函数                                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 创建按钮发送自定义事件
   ObjectCreate(0,buttonID,OBJ_BUTTON,0,100,100);
```

© 2000-2025, MetaQuotes Ltd.

```
   ObjectSetInteger(0,buttonID,OBJPROP_COLOR,clrWhite);
   ObjectSetInteger(0,buttonID,OBJPROP_BGCOLOR,clrGray);
   ObjectSetInteger(0,buttonID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_XSIZE,200);
   ObjectSetInteger(0,buttonID,OBJPROP_YSIZE,50);
   ObjectSetString(0,buttonID,OBJPROP_FONT,"Arial");
   ObjectSetString(0,buttonID,OBJPROP_TEXT,"Button");
   ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
   ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

//--- 创建标签展示信息
   ObjectCreate(0,labelID,OBJ_LABEL,0,100,100);
   ObjectSetInteger(0,labelID,OBJPROP_COLOR,clrRed);
   ObjectSetInteger(0,labelID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,labelID,OBJPROP_YDISTANCE,50);
   ObjectSetString(0,labelID,OBJPROP_FONT,"Trebuchet MS");
   ObjectSetString(0,labelID,OBJPROP_TEXT,"No information");
   ObjectSetInteger(0,labelID,OBJPROP_FONTSIZE,20);
   ObjectSetInteger(0,labelID,OBJPROP_SELECTABLE,0);

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| 专家无法初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ObjectDelete(0,buttonID);
   ObjectDelete(0,labelID);
  }
//+------------------------------------------------------------------+
//| 专家订单号函数                                                   |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- 通过按下鼠标键检测事件
   if(id==CHARTEVENT_OBJECT_CLICK)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       string clickedChartObject=sparam;
       //--- 如果点击带有按键ID名的物件
       if(clickedChartObject==buttonID)
         {
          //--- 按键状态-按或者不按
          bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
          //--- 调试信息日志
          Print("Button pressed = ",selected);
          int customEventID; // 发送的自定义事件数
          string message;    // 事件中被发送的信息
          //--- 如果按下按键
          if(selected)
            {
             message="Button pressed";
             customEventID=CHARTEVENT_CUSTOM+1;
            }
          else // 不按按键
            {
             message="Button in not pressed";
             customEventID=CHARTEVENT_CUSTOM+999;
            }
          //--- 发送自定义事件 "our"图表
          EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
          ///--- 发送信息到所有打开的图表
          BroadcastEvent(ChartID(),0,"Broadcast Message");
          //--- 调试信息
          Print("Sent an event with ID = ",customEventID);
         }
       ChartRedraw();// 强制重画所有图表物件
      }

//--- 检测属于用户事件的事件
   if(id>CHARTEVENT_CUSTOM)
     {
      if(id==broadcastEventID)
        {
         Print("Got broadcast message from a chart with id = "+lparam);
        }
      else
        {
         //--- 阅读事件中的文本信息
         string info=sparam;
         Print("Handle the user event with the ID = ",id);
         //--- 标签中展示信息
         ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
         ChartRedraw();// 强制重画所有图表物件
        }
     }
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 发送广播事件到所有打开的图表                                     |
//+------------------------------------------------------------------+
void BroadcastEvent(long lparam,double dparam,string sparam)
  {
   int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
   long currChart=ChartFirst();
   int i=0;
   while(i<CHARTS_MAX)                 // 打开图表不能多于CHARTS_MAX
     {
      EventChartCustom(currChart,eventID,lparam,dparam,sparam);
      currChart=ChartNext(currChart); // 从上一个接收新图表
      if(currChart==-1) break;        // 达到图表列表末端
      i++;                            // 不要忘记增加计数器
     }
  }
//+------------------------------------------------------------------+
```

另见

客户端事件，事件处理程序函数

© 2000-2025, MetaQuotes Ltd.

# 使用OpenCL工作

OpenCL程序用来执行支持OpenCL 1.1或更高版本视频卡上的计算. 现代视频卡包含数百个小型专用处理器，可以同时与输入数据流进行简单的数学运算. OpenCL语言组织了并行计算并且为一类特定任务提供了更快的速度.

在一些图形卡中默认禁止处理double类型数字。这可能会导致编译错误5105。若要启用支持double类型数字，请添加以下指令到您的OpenCL程序：#pragma OPENCL EXTENSION cl_khr_fp64 : enable。但是，如果图形卡不支持double型，启用该指令则不会有帮助。

建议以单独的CL文件编写OpenCL源代码，稍后将其包含进使用资源变量MQL5程序。

## 处理OpenCL程序中的错误

若要获取有关OpenCL程序中最后一个错误的信息，请使用允许获取错误代码和文本描述的CLGetInfoInteger和CLGetInfoString函数。

OpenCL最后错误代码：要获取最新OpenCL错误，请调用CLGetInfoInteger，同时忽略*handle*参数（可以设置为零）。错误描述：https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS。

对于未知错误代码，返回"unknown OpenCL errorN"字符串，这里N为错误代码。例如：

```
//--- 获取最后一个错误代码时忽略第一个'handle'参数
intcode= (int)CLGetInfoInteger(0,CL_LAST_ERROR);
```

OpenCL错误的文本描述：要获取最新OpenCL错误，请调用CLGetInfoString。错误代码应传递为*handle*参数。

错误描述：https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS。如果传递的是CL_LAST_ERROR而不是错误代码，那么函数将返回最后一个错误的描述。例如：

```
//--- 获取最后一个OpenCL错误的代码
int    code= (int)CLGetInfoInteger(0,CL_LAST_ERROR);
stringdesc;// 要获取错误文本描述

//--- 使用错误代码来获取错误文本描述
if(!CLGetInfoString(code,CL_ERROR_DESCRIPTION,desc))
 desc="无法获取OpenCL错误描述，"+ (string)GetLastError();
Print(desc);

//--- 若要在不先获取代码的情况下，获取最后一个OpenCL错误的描述，请传递CL_LAST_ERROR
if(!CLGetInfoString(CL_LAST_ERROR,CL_ERROR_DESCRIPTION,desc))
 desc="无法获取OpenCL错误描述，"+ (string)GetLastError();
Print(desc);;
```

到目前为止，作为错误描述，给出内部枚举的名称。您可以在这里找到它的解码：https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS。例如，CL_INVALID_KERNEL_ARGS值表示 "当某些内核参数未设置或无效时，在将内核排队时返回。"

运行OpenCL程序中的函数：

© 2000-2025, MetaQuotes Ltd.

| 函数 | 操作 |
| --- | --- |
| CLHandleType | 返回OpenCL处理类型作为 ENUM_OPENCL_HANDLE_TYPE 详表中的某个值. |
| CLGetInfoInteger | 为OpenCL对象或设备返回一个整数属性值 |
| CLContextCreate | 创建OpenCL背景 |
| CLContextFree | 移除OpenCL背景 |
| CLGetDeviceInfo | 从OpenCL驱动程序接收设备属性 |
| CLProgramCreate | 从源代码创建OpenCL程序 |
| CLProgramFree | 移除OpenCL程序 |
| CLKernelCreate | 创建OpenCL开始函数 |
| CLKernelFree | 移除OpenCL开始函数 |
| CLSetKernelArg | 为OpenCL函数设置一个参数 |
| CLSetKernelArgMem | 设置OpenCL缓冲作为OpenCL函数的参数 |
| CLSetKernelArgMemLocal | 设置本地缓冲区作为内核函数参数 |
| CLBufferCreate | 创建OpenCL缓冲 |
| CLBufferFree | 删除OpenCL缓冲 |
| CLBufferWrite | 写入数组到OpenCL缓冲 |
| CLBufferRead | 读取OpenCL缓冲到数组 |
| CLExecute | 运行OpenCL程序 |
| CLExecutionStatus | 返回OpenCL程序执行状态。 |

另见

OpenCL, 资源

© 2000-2025, MetaQuotes Ltd.

# CLHandleType

返回OpenCL处理类型作为 ENUM_OPENCL_HANDLE_TYPE 详表中的某个值.

```
ENUM_OPENCL_HANDLE_TYPE  CLHandleType(
   int  handle     // OpenCL对象处理
   );
```

参数

*handle*

[in] OpenCL对象处理:一个背景,一个内核或一个OpenCL程序.

返回值

作为 ENUM_OPENCL_HANDLE_TYPE 详表中的某个值的 OpenCL处理类型.

**ENUM_OPENCL_HANDLE_TYPE**

| 标识符 | 描述 |
| --- | --- |
| OPENCL_INVALID | 不正确处理 |
| OPENCL_CONTEXT | OpenCL背景处理 |
| OPENCL_PROGRAM | OpenCL程序处理 |
| OPENCL_KERNEL | OpenCL内核处理 |
| OPENCL_BUFFER | OpenCL缓冲处理 |

© 2000-2025, MetaQuotes Ltd.

# CLGetInfoInteger

为一个OpenCL对象或者设备返回整数属性的数值.

```
long  CLGetInfoInteger(
   int  handle,                              // OpenCL对象处理或OpenCL设备编号
   ENUM_OPENCL_PROPERTY_INTEGER  prop        // 被请求属性
   );
```

参数

*handle*

[in] OpenCL对象处理或OpenCL设备编号. OpenCL设备编号开始于零.

*prop*

[in] 从 ENUM_OPENCL_PROPERTY_STRING 详表被请求的属性类型, 您想要被包含的值.

返回值

如果成功属性的值或一旦错误 -1 . 有关错误信息, 使用 GetLastError() 函数.

ENUM_OPENCL_PROPERTY_INTEGER

| 标识符 | 描述 | 类型 |
|---|---|---|
| CL_DEVICE_COUNT | OpenCL支持的设备编号. 此属性不要求第一个参数规范, 即你可以处理参数通过一个为零的值 | int |
| CL_DEVICE_TYPE | 设备类型 | ENUM_CL_DEVICE_TYPE |
| CL_DEVICE_VENDOR_ID | 唯一的供应商识别 | uint |
| CL_DEVICE_MAX_COMPUTE_UNITS | OpenCL设备中的并行计算任务编号. 一个工作组解决了一个计算任务.最低值为1 | uint |
| CL_DEVICE_MAX_CLOCK_FREQUENCY | 设备最高的固定频率兆赫. | uint |
| CL_DEVICE_GLOBAL_MEM_SIZE | 设备的全局内存字节大小 | ulong |
| CL_DEVICE_LOCAL_MEM_SIZE | 处理数据 (事件)本地内存的字节大小 | uint |
| CL_BUFFER_SIZE | OpenCL字节缓冲区的实际大小 | ulong |
| CL_DEVICE_MAX_WORK_GROUP_SIZE | 可用于OpenCL设备的本地工作组总数。 | ulong |
| CL_KERNEL_WORK_GROUP_SIZE | 可用于OpenCL程序的本地工作组总数。 | ulong |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 | 类型 |
| --- | --- | --- |
| CL_KERNEL_LOCAL_MEM_SIZE | 由OpenCL程序用于解决一组全部并行任务的本地存储器大小（字节）。使用CL_DEVICE_LOCAL_MEM_SIZE接收最大可用值 | ulong |
| CL_KERNEL_PRIVATE_MEM_SIZE | OpenCL程序内核中，每个任务使用的最小专有存储器大小（字节）。 | ulong |
| CL_LAST_ERROR | 最后一个OpenCL错误的值 | int |

ENUM_CL_DEVICE_TYPE详表包含可以支持OpenCL的设备类型。您可以通过CLGetInfoInteger(handle_or_deviceN, CL_DEVICE_TYPE)用其编号或者OpenCL处理对象接收设备类型。

**ENUM_CL_DEVICE_TYPE**

| 标识符 | 描述 |
| --- | --- |
| CL_DEVICE_ACCELERATOR | OpenCL专用加速器（比如，IBM CELL Blade)。这些设备使用外设互连如PCIe与主处理器进行通信。 |
| CL_DEVICE_CPU | 一个OpenCL主机处理器设备。主机处理器运行OpenCL运转且为一个单核或多核CPU。 |
| CL_DEVICE_GPU | 一个OpenCL设备是GPU。 |
| CL_DEVICE_DEFAULT | 系统默认的OpenCL设备。默认设备不能为CL_DEVICE_TYPE_CUSTOM 设备。 |
| CL_DEVICE_CUSTOM | 专用的加速器不支持OpenCL C编写的程序。 |

例如：

```
void OnStart()
  {
   int cl_ctx;
//--- 初始化 OpenCL 上下文
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
//--- 显示有关OpenCL设备的常规信息
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY)," MHz"
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE)," by
```

© 2000-2025, MetaQuotes Ltd.

```
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE)," byte
//--- free OpenCL context
   CLContextFree(cl_ctx);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CLGetInfoString

为OpenCL对象或者设备返回一个属性的字符串值.

```
bool  CLGetInfoString(
   int  handle,                              // OpenCL对象处理或者OpenCL设备号码
   ENUM_OPENCL_PROPERTY_STRING  prop,        // 被请求的属性
   string&  value                            // 引用字符串
   );
```

参数

*handle*

[in] OpenCL对象处理或者OpenCL设备号码. OpenCL设备编号开始于零.

*prop*

[in] 从 ENUM_OPENCL_PROPERTY_STRING 详表被请求的属性类型, 应当被包含的值.

*value*

[out] 接收属性值的字符串.

返回值

成功则返回true, 否则返回 false. 有关错误信息, 使用 GetLastError() 函数.

**ENUM_OPENCL_PROPERTY_STRING**

| 标识符 | 描述 |
|---|---|
| CL_PLATFORM_PROFILE | CL_PLATFORM_PROFILE - OpenCL简介. 简介名称可以是以下值中的某一个:<br>• FULL_PROFILE - 实现支持OpenCL 功能定义为核心规范的一部分而无需为OpenCL支持额外的扩展);<br>• EMBEDDED_PROFILE - 实现支持OpenCL作为增补. 修订简介被定义为每个OpenCL版本的子集. |
| CL_PLATFORM_VERSION | OpenCL版本 |
| CL_PLATFORM_VENDOR | 平台供应商名称 |
| CL_PLATFORM_EXTENSIONS | 由平台支持的扩展名列表。扩展名应该能被所有与这个平台相 |
| CL_DEVICE_NAME | 设备名称 |
| CL_DEVICE_VENDOR | 供应商名称 |
| CL_DRIVER_VERSION | major_number.minor_number格式OpenCL设备版本 |
| CL_DEVICE_PROFILE | OpenCL设备简介. 简介名称可以是以下值的某一个: |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 描述 |
| --- | --- |
| | • FULL_PROFILE - 实现支持OpenCL 功能定义为核心规范的一部分而无需为OpenCL支持额外的扩展);<br>• EMBEDDED_PROFILE -实现支持OpenCL作为增补. 修订简介被定义为每个OpenCL版本的子集. |
| CL_DEVICE_VERSION | "OpenCL<space><major_version.minor_version><space><vendor-specific information>" 格式OpenCL版本 |
| CL_DEVICE_EXTENSIONS | 设备所支持的扩展名列表。列表可以包含由供应商支持的扩展，以及一个或多个批准名称:<br>cl_khr_int64_base_atomics<br>cl_khr_int64_extended_atomics<br>cl_khr_fp16<br>cl_khr_gl_sharing<br>cl_khr_gl_event<br>cl_khr_d3d10_sharing<br>cl_khr_dx9_media_sharing<br>cl_khr_d3d11_sharing |
| CL_DEVICE_BUILT_IN_KERNELS | 可支持的内核列表用 ";"分割. |
| CL_DEVICE_OPENCL_C_VERSION | 这个设备的编译器所支持的最大版本。版本格式:<br>"OpenCL<space>C<space><major_version.minor_version><space><vendor-specific information> " |
| CL_ERROR_DESCRIPTION | OpenCL错误的文本描述 |

例如:

```
void OnStart()
  {
   int cl_ctx;
   string str;
//--- 初始化 OpenCL 上下文
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
//--- 显示平台信息
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_NAME,str))
      Print("OpenCL platform name: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VENDOR,str))
      Print("OpenCL platform vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VERSION,str))
      Print("OpenCL platform ver: ",str);
```

© 2000-2025, MetaQuotes Ltd.

```
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_PROFILE,str))
      Print("OpenCL platform profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_EXTENSIONS,str))
      Print("OpenCL platform ext: ",str);
//--- 显示设备信息
   if(CLGetInfoString(cl_ctx,CL_DEVICE_NAME,str))
      Print("OpenCL device name: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_PROFILE,str))
      Print("OpenCL device profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_BUILT_IN_KERNELS,str))
      Print("OpenCL device kernels: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_EXTENSIONS,str))
      Print("OpenCL device ext: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VENDOR,str))
      Print("OpenCL device vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VERSION,str))
      Print("OpenCL device ver: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_OPENCL_C_VERSION,str))
      Print("OpenCL open c ver: ",str);
//--- 显示OpenCL设备的常规信息
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY));
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE));
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE));
//--- free OpenCL context
   CLContextFree(cl_ctx);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CLContextCreate

创建一个OpenCL背景并返回其处理.

```
int  CLContextCreate(
   int  device=CL_USE_ANY     // OpenCL设备或宏的序列号
   );
```

### Parameter

*device*

[in] 系统中的OpenCL设备序号. 你可以指定下列值之一，而不是一个具体的数字:

- CL_USE_ANY -任何OpenCL支持的有效设备均被允许;
- CL_USE_CPU_ONLY -仅CPU中的OpenCL仿真被允许;
- CL_USE_GPU_ONLY -OpenCL仿真被禁止并且仅有OpenCL支持的专业设备（视频卡）可以被使用;
- CL_USE_GPU_DOUBLE_ONLY - 只允许支持double类型的GPU。

### Return Value

如果成功将处理一个OpenCL对象。一旦出错 -返回1. 有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLContextFree

删除一个OpenCL背景.

```
void  CLContextFree(
   int  context     // 处理一个OpenCL背景
   );
```

参数

*context*

[in] OpenCL背景处理.

返回值

无. 在内部错误情况下 _LastError 的值会改变. 有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLGetDeviceInfo

函数从OpenCL驱动器接收设备的性能.

```
bool  CLGetDeviceInfo(
   int      handle,          // OpenCL设备处理
   int      property_id,     // 被请求的属性 ID
   uchar&   data[],          // 数据接收数组
   uint&    size             // 数组中的元素转换，默认为0
   );
```

参数

*handle*

[in] 由 CLContextCreate() 函数创建OpenCL设备索引或OpenCL处理.

*property_id*

[in] OpenCL设备属性 ID 应当被接收. 该值是列于 table below 中可以被预定的某个值.

*data[]*

[out] 用于接收数据的数组具有被要求的属性.

*size*

[out] 接收数据的数组 *data[]* 大小.

返回值

如果成功则为true，否则为 false. 有关错误信息，使用 GetLastError() 函数.

注意

对于一维数组，考虑到 AS_SERIES 旗帜，元素的数量，从写入读取数据到一个OpenCL缓冲开始处计算.

可用的OpenCL设备性能ID列表

性能的准确描述及其函数可以在 the official OpenCL web site 找到.

| 标识符 | 值 |
|---|---|
| CL_DEVICE_TYPE | 0x1000 |
| CL_DEVICE_VENDOR_ID | 0x1001 |
| CL_DEVICE_MAX_COMPUTE_UNITS | 0x1002 |
| CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS | 0x1003 |
| CL_DEVICE_MAX_WORK_GROUP_SIZE | 0x1004 |
| CL_DEVICE_MAX_WORK_ITEM_SIZES | 0x1005 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR | 0x1006 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 值 |
| --- | --- |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT | 0x1007 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT | 0x1008 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG | 0x1009 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT | 0x100A |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE | 0x100B |
| CL_DEVICE_MAX_CLOCK_FREQUENCY | 0x100C |
| CL_DEVICE_ADDRESS_BITS | 0x100D |
| CL_DEVICE_MAX_READ_IMAGE_ARGS | 0x100E |
| CL_DEVICE_MAX_WRITE_IMAGE_ARGS | 0x100F |
| CL_DEVICE_MAX_MEM_ALLOC_SIZE | 0x1010 |
| CL_DEVICE_IMAGE2D_MAX_WIDTH | 0x1011 |
| CL_DEVICE_IMAGE2D_MAX_HEIGHT | 0x1012 |
| CL_DEVICE_IMAGE3D_MAX_WIDTH | 0x1013 |
| CL_DEVICE_IMAGE3D_MAX_HEIGHT | 0x1014 |
| CL_DEVICE_IMAGE3D_MAX_DEPTH | 0x1015 |
| CL_DEVICE_IMAGE_SUPPORT | 0x1016 |
| CL_DEVICE_MAX_PARAMETER_SIZE | 0x1017 |
| CL_DEVICE_MAX_SAMPLERS | 0x1018 |
| CL_DEVICE_MEM_BASE_ADDR_ALIGN | 0x1019 |
| CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE | 0x101A |
| CL_DEVICE_SINGLE_FP_CONFIG | 0x101B |
| CL_DEVICE_GLOBAL_MEM_CACHE_TYPE | 0x101C |
| CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE | 0x101D |
| CL_DEVICE_GLOBAL_MEM_CACHE_SIZE | 0x101E |
| CL_DEVICE_GLOBAL_MEM_SIZE | 0x101F |
| CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE | 0x1020 |
| CL_DEVICE_MAX_CONSTANT_ARGS | 0x1021 |
| CL_DEVICE_LOCAL_MEM_TYPE | 0x1022 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 值 |
| --- | --- |
| CL_DEVICE_LOCAL_MEM_SIZE | 0x1023 |
| CL_DEVICE_ERROR_CORRECTION_SUPPORT | 0x1024 |
| CL_DEVICE_PROFILING_TIMER_RESOLUTION | 0x1025 |
| CL_DEVICE_ENDIAN_LITTLE | 0x1026 |
| CL_DEVICE_AVAILABLE | 0x1027 |
| CL_DEVICE_COMPILER_AVAILABLE | 0x1028 |
| CL_DEVICE_EXECUTION_CAPABILITIES | 0x1029 |
| CL_DEVICE_QUEUE_PROPERTIES | 0x102A |
| CL_DEVICE_NAME | 0x102B |
| CL_DEVICE_VENDOR | 0x102C |
| CL_DRIVER_VERSION | 0x102D |
| CL_DEVICE_PROFILE | 0x102E |
| CL_DEVICE_VERSION | 0x102F |
| CL_DEVICE_EXTENSIONS | 0x1030 |
| CL_DEVICE_PLATFORM | 0x1031 |
| CL_DEVICE_DOUBLE_FP_CONFIG | 0x1032 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF | 0x1034 |
| CL_DEVICE_HOST_UNIFIED_MEMORY | 0x1035 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR | 0x1036 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_SHORT | 0x1037 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_INT | 0x1038 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG | 0x1039 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT | 0x103A |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE | 0x103B |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF | 0x103C |
| CL_DEVICE_OPENCL_C_VERSION | 0x103D |
| CL_DEVICE_LINKER_AVAILABLE | 0x103E |
| CL_DEVICE_BUILT_IN_KERNELS | 0x103F |
| CL_DEVICE_IMAGE_MAX_BUFFER_SIZE | 0x1040 |
| CL_DEVICE_IMAGE_MAX_ARRAY_SIZE | 0x1041 |

© 2000-2025, MetaQuotes Ltd.

| 标识符 | 值 |
| --- | --- |
| CL_DEVICE_PARENT_DEVICE | 0x1042 |
| CL_DEVICE_PARTITION_MAX_SUB_DEVICES | 0x1043 |
| CL_DEVICE_PARTITION_PROPERTIES | 0x1044 |
| CL_DEVICE_PARTITION_AFFINITY_DOMAIN | 0x1045 |
| CL_DEVICE_PARTITION_TYPE | 0x1046 |
| CL_DEVICE_REFERENCE_COUNT | 0x1047 |
| CL_DEVICE_PREFERRED_INTEROP_USER_SYNC | 0x1048 |
| CL_DEVICE_PRINTF_BUFFER_SIZE | 0x1049 |
| CL_DEVICE_IMAGE_PITCH_ALIGNMENT | 0x104A |
| CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT | 0x104B |

例如：

```
void OnStart()
  {
//---
   int dCount= CLGetInfoInteger(0,CL_DEVICE_COUNT);
   for(int i = 0; i<dCount; i++)
     {
      int clCtx=CLContextCreate(i);
      if(clCtx == -1)
         Print("ERROR in CLContextCreate");
      string device;
      CLGetInfoString(clCtx,CL_DEVICE_NAME,device);
      Print(i,": ",device);
      uchar data[1024];
      uint size;
      CLGetDeviceInfo(clCtx,CL_DEVICE_VENDOR,data,size);
      Print("size = ",size);
      string str=CharArrayToString(data);
      Print(str);
     }
  }
//--- example of entries in Experts journal
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      2: Advanced Micro Devices, Inc.
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      size = 32
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      Tahiti
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      Intel(R) Corporation
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      size = 21
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      1:        Intel(R) Core(TM) i7-377
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      NVIDIA Corporation
```

© 2000-2025, MetaQuotes Ltd.

```
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      size = 19
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      0: GeForce GTX 580
```

© 2000-2025, MetaQuotes Ltd.

# CLProgramCreate

从源代码创建一个OpenCL程序.

```
int  CLProgramCreate(
   int           context,     // 处理一个OpenCL背景
   const string  source       // 源代码
   );
```

参数

*context*

[in] 处理一个OpenCL背景.

*source*

[in]OpenCL程序源代码字符串.

返回值

如果成功将处理一个OpenCL对象。一旦出错 -返回1 . 有关错误信息，使用 GetLastError() 函数 .

注意

目前，以下错误代码正在使用:

- ERR_OPENCL_INVALID_HANDLE -OpenCL上下文的无效处理程序。
- ERR_INVALID_PARAMETER -无效字符串参数。
- ERR_NOT_ENOUGH_MEMORY -完成操作内存不足。
- ERR_OPENCL_PROGRAM_CREATE -OpenCL 内部错误或编译错误。

在一些图形卡中，默认禁止启用双精度 类型数字。这可能导致编译错误5105。若要启用支持双精度类型数字，请添加以下指令到您的OpenCL程序：#pragma OPENCL EXTENSION cl_khr_fp64 : enable

示例：

```
//+------------------------------------------------------------------+
//| OpenCL kernel                                                    |
//+------------------------------------------------------------------+
const string
cl_src=
       //--- 默认一些 GPU 不支持双精度类型
       //--- cl_khr_fp64 指令用于启用处理双精度类型
       "#pragma OPENCL EXTENSION cl_khr_fp64 : enable      \r\n"
       //--- OpenCL 核函数
       "__kernel void Test_GPU(__global double *data,      \r\n"
       "                       const    int N,             \r\n"
       "                       const    int total_arrays)  \r\n"
       "  {                                                 \r\n"
       "   uint kernel_index=get_global_id(0);              \r\n"
       "   if (kernel_index>total_arrays) return;           \r\n"
       "   uint local_start_offset=kernel_index*N;          \r\n"
```

© 2000-2025, MetaQuotes Ltd.

```
        "   for(int i=0; i<N; i++)                                \r\n"
        "     {                                                   \r\n"
        "       data[i+local_start_offset] *= 2.0;                \r\n"
        "     }                                                   \r\n"
        "  }                                                      \r\n";
//+------------------------------------------------------------------+
//| Test_CPU                                                         |
//+------------------------------------------------------------------+
bool Test_CPU(double &data[],const int N,const int id,const int total_arrays)
  {
//--- 检查数组大小
   if(ArraySize(data)==0) return(false);
//--- 检查数组索引
   if(id>total_arrays) return(false);
//--- 通过索引id计算数组局部偏移
   int local_start_offset=id*N;
//--- 元素乘以2
   for(int i=0; i<N; i++)
     {
      data[i+local_start_offset]*=2.0;
     }
   return true;
  }
//---
#define ARRAY_SIZE   100  // size of the array
#define TOTAL_ARRAYS 5    // total arrays
//--- OpenCL 处理程序
int cl_ctx;  // OpenCL 内容处理
int cl_prg;  // OpenCL 程序处理
int cl_krn;  // OpenCL 核处理
int cl_mem;  // OpenCL 缓冲处理
//---
double DataArray1[]; // 用于CPU计算的数据数组
double DataArray2[]; // 用于GPU计算的数据数组
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart()
  {
//--- 初始化OpenCL对象
//--- 创建OpenCL 环境
   if((cl_ctx=CLContextCreate())==INVALID_HANDLE)
     {
      Print("OpenCL not found. Error=",GetLastError());
      return(1);
     }
//--- 创建 OpenCL 程序
   if((cl_prg=CLProgramCreate(cl_ctx,cl_src))==INVALID_HANDLE)
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      CLContextFree(cl_ctx);
      Print("OpenCL program create failed. Error=",GetLastError());
      return(1);
     }
//--- 创建OpenCL 内核
   if((cl_krn=CLKernelCreate(cl_prg,"Test_GPU"))==INVALID_HANDLE)
     {
      CLProgramFree(cl_prg);
      CLContextFree(cl_ctx);
      Print("OpenCL kernel create failed. Error=",GetLastError());
      return(1);
     }
//--- 创建 OpenCL 缓冲
   if((cl_mem=CLBufferCreate(cl_ctx,ARRAY_SIZE*TOTAL_ARRAYS*sizeof(double),CL_MEM_REAI
     {
      CLKernelFree(cl_krn);
      CLProgramFree(cl_prg);
      CLContextFree(cl_ctx);
      Print("OpenCL buffer create failed. Error=",GetLastError());
      return(1);
     }
//--- 设置OpenCL 内核常量参数
   CLSetKernelArgMem(cl_krn,0,cl_mem);
   CLSetKernelArg(cl_krn,1,ARRAY_SIZE);
   CLSetKernelArg(cl_krn,2,TOTAL_ARRAYS);
//--- 准备数据数组
   ArrayResize(DataArray1,ARRAY_SIZE*TOTAL_ARRAYS);
   ArrayResize(DataArray2,ARRAY_SIZE*TOTAL_ARRAYS);
//--- 用数据填写数组
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- 计算 jth 数组局部开始偏移
      uint local_offset=j*ARRAY_SIZE;
      //--- 准备 j 索引数组
      for(int i=0; i<ARRAY_SIZE; i++)
        {
         //--- 用MathCos(i+j)函数填写数组;
         DataArray1[i+local_offset]=MathCos(i+j);
         DataArray2[i+local_offset]=MathCos(i+j);
        }
     };
//--- 测试CPU 计算
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- 计算 j 索引数组
      Test_CPU(DataArray1,ARRAY_SIZE,j,TOTAL_ARRAYS);
     }
//--- 准备 CLExecute 参数
   uint  offset[]={0};
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 全局工作大小
   uint  work[]={TOTAL_ARRAYS};
//--- 编写数据到 OpenCL 缓冲
   CLBufferWrite(cl_mem,DataArray2);
//--- 执行OpenCL 内核
   CLExecute(cl_krn,1,offset,work);
//--- 从OpenCL缓冲阅读数据
   CLBufferRead(cl_mem,DataArray2);
//--- 总误差
   double total_error=0;
//--- 比较结果和计算误差
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- 计算 jth 数组局部偏移
      uint local_offset=j*ARRAY_SIZE;
      //--- 比较结果
      for(int i=0; i<ARRAY_SIZE; i++)
        {
         double v1=DataArray1[i+local_offset];
         double v2=DataArray2[i+local_offset];
         double delta=MathAbs(v2-v1);
         total_error+=delta;
         //--- 显示初始和最后的数组
         if((j==0) || (j==TOTAL_ARRAYS-1))
            PrintFormat("array %d of %d, element [%d]:  %f, %f, [error]=%f",j+1,TOTAL_
        }
     }
   PrintFormat("Total error: %f",total_error);
//--- 删除OpenCL 对象
//--- 释放 OpenCL 缓冲
   CLBufferFree(cl_mem);
//--- 释放 OpenCL 内核
   CLKernelFree(cl_krn);
//--- 释放 OpenCL 程序
   CLProgramFree(cl_prg);
//--- 释放 OpenCL 环境
   CLContextFree(cl_ctx);
//---
   return(0);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CLProgramFree

删除一个OpenCL程序.

```
void  CLProgramFree(
   int  program     // 处理一个OpenCL对象
   );
```

参数

*program*

[in] OpenCL对象处理.

返回值

无. 在内部错误情况下 _LastError 的值会改变. 有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLKernelCreate

创建OpenCL程序内核并返回其处理.

```
int  CLKernelCreate(
   int           program,        // 处理一个OpenCL对象
   const string  kernel_name     // 内核名称
   );
```

参数

*program*

[in] 处理一个OpenCL程序对象.

*kernel_name*

[in] 在执行适当的OpenCL程序开始的地方的函数名称.

返回值

如果成功将处理一个OpenCL对象。一旦出错 -返回1 . 有关错误信息，使用 GetLastError() 函数 .

注意

目前，以下错误代码正在使用:

- ERR_OPENCL_INVALID_HANDLE - *program* OpenCL的无效处理程序。
- ERR_INVALID_PARAMETER - 无效字符串参数。
- ERR_OPENCL_TOO_LONG_KERNEL_NAME - 内核名称包含超过127个字符。
- ERR_OPENCL_KERNEL_CREATE - 创建OpenCL对象时发生的内部错误。

© 2000-2025, MetaQuotes Ltd.

# CLKernelFree

删除一个OpenCL启动函数.

```
void  CLKernelFree(
   int  kernel     // 处理一个OpenCL程序内核
   );
```

参数

*kernel_name*

[in] 内核对象处理.

返回值

无. 在内部错误情况下 _LastError 的值会改变. 有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLSetKernelArg

为OpenCL函数设置一个参数.

```
bool  CLSetKernelArg(
   int   kernel,         // 处理一个OpenCL程序内核
   uint  arg_index,      // OpenCL函数幅角的数量
   void  arg_value       // 源代码
   );
```

参数

*kernel*

[in] 处理一个OpenCL程序内核.

*arg_index*

[in] 函数幅角的数量，计数开始于零.

*arg_value*

[in] 函数幅角的值.

返回值

如果成功则返回true，否则返回false。有关错误信息，使用 GetLastError() 函数.

注意

目前，以下错误代码正在使用：

- ERR_INVALID_PARAMETER,
- ERR_OPENCL_INVALID_HANDLE - invalid handle to the OpenCL kernel.
- ERR_OPENCL_SET_KERNEL_PARAMETER - internal error of OpenCL.

© 2000-2025, MetaQuotes Ltd.

# CLSetKernelArgMem

设置一个OpenCL缓冲作为OpenCL函数的参数.

```
bool  CLSetKernelArgMem(
   int   kernel,           // 处理一个OpenCL程序的内核
   uint  arg_index,        // OpenCL函数参数的数量
   int   cl_mem_handle     // 处理一个OpenCL缓冲
   );
```

参数

*kernel*

[in] 处理一个OpenCL程序的内核.

*arg_index*

[in] 该函数参数的数量，编号起始于零.

*cl_mem_handle*

[in] 一个OpenCL缓冲处理.

返回值

如果成功则返回true，否则返回false。有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLSetKernelArgMemLocal

设置本地缓冲区作为内核函数参数。

```
bool  CLSetKernelArgMemLocal(
   int    kernel,           // 处理OpenCL程序的内核
   uint   arg_index,        // OpenCL函数参数的数值
   ulong  local_mem_size    // 缓冲区大小
   );
```

参数

*kernel*

[in] 处理OpenCL程序的内核。

*arg_index*

[in] 函数参数的数值，从0开始计数。

*local_mem_size*

[in] 缓冲区的字节大小。

返回值

如果成功返回true，否则返回false。如果出现错误信息，请使用GetLastError()函数。

© 2000-2025, MetaQuotes Ltd.

# CLBufferCreate

创建一个OpenCL缓冲并返回其操作.

```
int  CLBufferCreate(
   int   context,     // 处理一个OpenCL背景
   uint  size,        // 缓冲大小
   uint  flags        // 指定OpenCL缓冲性能的标志组合
   );
```

参数

*context*

[in] 一个OpenCL背景处理.

*size*

[in] 缓冲字节大小.

*flags*

[in] 使用标志性组合的缓冲性能:CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY, CL_MEM_ALLOC_HOST_PTR.

返回值

OpenCL缓冲处理成功.一旦出错 -返回1.有关错误信息,使用 GetLastError() 函数.

注意

目前,以下错误代码正在使用:

- ERR_OPENCL_INVALID_HANDLE -OpenCL背景的无效处理.
- ERR_NOT_ENOUGH_MEMORY -内存不足.
- ERR_OPENCL_BUFFER_CREATE -创建缓冲区的内部错误.

© 2000-2025, MetaQuotes Ltd.

# CLBufferFree

删除OpenCL缓冲.

```
void  CLBufferFree(
   int   buffer     // 处理一个OpenCL缓冲
   );
```

参数

*buffer*

[in] 一个OpenCL缓冲处理.

返回值

无. 在内部错误情况下 _LastError 的值会改变. 有关错误信息，使用 GetLastError() 函数.

© 2000-2025, MetaQuotes Ltd.

# CLBufferWrite

写入OpenCL缓冲区以及返回写入元素的个数.

```
uint  CLBufferWrite(
   int           buffer,                  // 处理一个OpenCL缓冲
   const void&   data[],                  // 值的数组
   uint          buffer_offset=0,         // 字节区偏移的OpenCL缓冲，默认为0
   uint          data_offset=0,           // 元素中数组的偏移，默认为0
   uint          data_count=WHOLE_ARRAY   // 从阵列写入的值的数量，默认整个阵列
   );
```

这里还有处理矩阵和向量的版本。

将矩阵中的值写入缓冲区，如果成功则返回true。

```
uint  CLBufferWrite(
   int           buffer,                  // OpenCL缓冲区的句柄
   uint          buffer_offset,           // 以字节为单位的OpenCL缓冲区抵消
   matrix<T>     &mat                     // 用于写入缓冲区的值矩阵
   );
```

将向量中的值写入缓冲区，如果成功则返回true。

```
uint  CLBufferWrite(
   int           buffer,                  // OpenCL缓冲区的句柄
   uint          buffer_offset,           // 以字节为单位的OpenCL缓冲区抵消
   vector<T>     &vec                     // 用于写入缓冲区的值向量
   );
```

参数

*buffer*

[in] 一个OpenCL缓冲处理.

*data[]*

[in] 一个用于从OpenCL的缓冲区接收的值的数组。通过引用传递.

*buffer_offset*

[in] 一个在字节区偏移的OpenCL缓冲，从读取开始处。默认情况下，读取开始于正开头的缓冲.

*data_offset*

[in] 第一个数组元素的索引，开始于写入OpenCL缓冲数组的值. 默认情况下，取数组正开头的值.

*data_count*

[in] 应被写入值的数量. 默认数组的所有值.

*mat*

[out] 用于将数据写入缓冲区的矩阵可以是三种类型中的任何一种 — matrix、matrixf或matrixc.

© 2000-2025, MetaQuotes Ltd.

*vec*

[out] 用于将数据写入缓冲区的向量可以是三种类型中的任何一种 — vector、vectorf或vectorc.

返回值

写入元素的数量. 错误情况下返回0. 有关错误信息，使用 GetLastError() 函数.

如果矩阵或向量成功处理，返回true，否则返回false。

注意

对于一维数组，考虑到 AS_SERIES 旗帜，元素的数量，从写入读取数据到一个OpenCL缓冲开始处计算.

两个或两个以上的尺寸数组是一维的。在这种情况下，*data_offset* 是要在演示文稿中跳过的元素个数，而不是在第一维的元素个数.

使用MatMul方法的矩阵乘法和OpenCL中并行计算的示例

```
#define M       3000      // 第一个矩阵中的行数
#define K       2000      // 第一个矩阵中的列数等于第二个矩阵的行数
#define N       3000      // 第二个矩阵中的列数

//+------------------------------------------------------------------+
const string clSrc=
  "#define N     "+IntegerToString(N)+"                              \r\n"
  "#define K     "+IntegerToString(K)+"                              \r\n"
  "                                                                 \r\n"
  "__kernel void matricesMul( __global float *in1,                  \r\n"
  "                           __global float *in2,                  \r\n"
  "                           __global float *out  )                \r\n"
  "{                                                                \r\n"
  "  int m = get_global_id( 0 );                                    \r\n"
  "  int n = get_global_id( 1 );                                    \r\n"
  "  float sum = 0.0;                                               \r\n"
  "  for( int k = 0; k < K; k ++ )                                  \r\n"
  "     sum += in1[ m * K + k ] * in2[ k * N + n ];                 \r\n"
  "  out[ m * N + n ] = sum;                                        \r\n"
  "}                                                                \r\n";
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- 初始化随机数生成器
  MathSrand((int)TimeCurrent());
//--- 用随机值填写给定大小的矩阵
  matrixf mat1(M, K, MatrixRandom) ;     // 第一个矩阵
  matrixf mat2(K, N, MatrixRandom);      //第二个矩阵
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 使用原始方法计算矩阵的乘积
  uint start=GetTickCount();
  matrixf matrix_naive=matrixf::Zeros(M, N);// 在这里设置两个矩阵相乘的结果
  for(int m=0; m<M; m++)
    for(int k=0; k<K; k++)
      for(int n=0; n<N; n++)
        matrix_naive[m][n]+=mat1[m][k]*mat2[k][n];
  uint time_naive=GetTickCount()-start;

//--- 通过MatMul计算矩阵的乘积
  start=GetTickCount();
  matrixf matrix_matmul=mat1.MatMul(mat2);
  uint time_matmul=GetTickCount()-start;

//--- 在OpenCL中计算矩阵的乘积
  matrixf matrix_opencl=matrixf::Zeros(M, N);
  int cl_ctx;              // 环境句柄
  if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
   {
    Print("OpenCL not found, leaving");
    return;
   }
  int cl_prg;              // 程序句柄
  int cl_krn;              // 内核句柄
  int cl_mem_in1;          // 第一个 输入)缓冲区句柄
  int cl_mem_in2;          // 第二个 输入)缓冲区句柄
  int cl_mem_out;          // 第三个 输出)缓冲区句柄
//--- 创建程序和内核
  cl_prg = CLProgramCreate(cl_ctx, clSrc);
  cl_krn = CLKernelCreate(cl_prg, "matricesMul");
//--- 为三个矩阵创建所有三个缓冲区
  cl_mem_in1=CLBufferCreate(cl_ctx, M*K*sizeof(float), CL_MEM_READ_WRITE);
  cl_mem_in2=CLBufferCreate(cl_ctx, K*N*sizeof(float), CL_MEM_READ_WRITE);
//--- 第三个矩阵 - 输出
  cl_mem_out=CLBufferCreate(cl_ctx, M*N*sizeof(float), CL_MEM_READ_WRITE);
//--- 设置内核参数
  CLSetKernelArgMem(cl_krn, 0, cl_mem_in1);
  CLSetKernelArgMem(cl_krn, 1, cl_mem_in2);
  CLSetKernelArgMem(cl_krn, 2, cl_mem_out);
//--- 将矩阵写入设备缓冲区
  CLBufferWrite(cl_mem_in1, 0, mat1);
  CLBufferWrite(cl_mem_in2, 0, mat2);
  CLBufferWrite(cl_mem_out, 0, matrix_opencl);
//--- OpenCL代码执行时间开始
  start=GetTickCount();
//--- 设置任务工作区参数，并执行OpenCL程序
  uint  offs[2] = {0, 0};
  uint works[2] = {M, N};
```

© 2000-2025, MetaQuotes Ltd.

```
   start=GetTickCount();
   bool ex=CLExecute(cl_krn, 2, offs, works);
//--- 计算矩阵结果
   if(CLBufferRead(cl_mem_out, 0, matrix_opencl))
     PrintFormat("[%d x %d] matrix read: ", matrix_opencl.Rows(), matrix_opencl.Cols())
    else
       Print("CLBufferRead(cl_mem_out, 0, matrix_opencl failed. Error ",GetLastError())
   uint time_opencl=GetTickCount()-start;
   Print("Compare calculation time using each method");
   PrintFormat("Naive product time = %d ms",time_naive);
   PrintFormat("MatMul product time = %d ms",time_matmul);
   PrintFormat("OpenCl product time = %d ms",time_opencl);
//--- 释放所有OpenCL环境
   CLFreeAll(cl_ctx, cl_prg, cl_krn, cl_mem_in1, cl_mem_in2, cl_mem_out);

//--- 比较所有获得的结果矩阵
   Print("How many discrepancy errors are there between result matrices?");
   ulong errors=matrix_naive.Compare(matrix_matmul,(float)1e-12);
   Print("matrix_direct.Compare(matrix_matmul,1e-12)=",errors);
   errors=matrix_matmul.Compare(matrix_opencl,float(1e-12));
   Print("matrix_matmul.Compare(matrix_opencl,1e-12)=",errors);
/*
   结果：

    [3000 x 3000] matrix read:
   比较每种方法的计算时间
    Naive product time = 54750 ms
    MatMul product time = 4578 ms
    OpenCl product time = 922 ms
   结果矩阵之间有多少误差？
    matrix_direct.Compare(matrix_matmul,1e-12)=0
    matrix_matmul.Compare(matrix_opencl,1e-12)=0
*/
 }
//+------------------------------------------------------------------+
//| 用随机值填写矩阵                                                 |
//+------------------------------------------------------------------+
void MatrixRandom(matrixf& m)
 {
  for(ulong r=0; r<m.Rows(); r++)
    {
     for(ulong c=0; c<m.Cols(); c++)
      {
       m[r][c]=(float)((MathRand()-16383.5)/32767.);
      }
    }
 }
//+------------------------------------------------------------------+
//| 释放所有OpenCL环境                                               |
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
void CLFreeAll(int cl_ctx, int cl_prg, int cl_krn,
               int cl_mem_in1, int cl_mem_in2, int cl_mem_out)
 {
//--- 以相反顺序删除OpenCL创建的所有环境
  CLBufferFree(cl_mem_in1);
  CLBufferFree(cl_mem_in2);
  CLBufferFree(cl_mem_out);
  CLKernelFree(cl_krn);
  CLProgramFree(cl_prg);
  CLContextFree(cl_ctx);
 }
```

© 2000-2025, MetaQuotes Ltd.

# CLBufferRead

读取一个OpenCL缓冲到一个数组并且返回读元件的数量.

```
uint  CLBufferRead(
   int           buffer,                  // 处理一个OpenCL缓冲
   const void&   data[],                  // 一个值的数组
   uint          buffer_offset=0,         // 在字节缓冲区偏移的OpenCL，默认为0
   uint          data_offset=0,           // 在元素的数组的偏移量，默认为0
   uint          data_count=WHOLE_ARRAY   // 用于读取的缓冲值的个数，默认为整个缓冲区
   );
```

这里还有处理矩阵和向量的版本。

读取OpenCL缓冲区到矩阵，如果成功则返回true。

```
uint  CLBufferRead(
   int           buffer,                  // OpenCL缓冲区的句柄
   uint          buffer_offset,           // 以字节为单位的OpenCL缓冲区抵消
   const matrxi& mat,                     // 用于接收缓冲区值的矩阵
   ulong         rows=-1,                 // 矩阵中的行数
   ulong         cols=-1                  // 矩阵中的列数
   );
```

读取OpenCL缓冲区到向量，如果成功则返回true。

```
uint  CLBufferRead(
   int           buffer,                  // OpenCL缓冲区的句柄
   uint          buffer_offset,           // 以字节为单位的OpenCL缓冲区抵消
   const vector& vec,                     // 用于接收缓冲区值的向量
   ulong         size-1,                  // 向量长度
   );
```

参数

*buffer*

[in] 一个OpenCL缓冲控制.

*data[]*

[in] 一个用于从OpenCL的缓冲区接收的值的数组。通过引用传递.

*buffer_offset*

[in] 一个在字节区偏移的OpenCL缓冲，从读取开始处。默认情况下，读取开始于正开头的缓冲.

*data_offset*

[in] 用于书写的OpenCL缓冲的值的第一个数组元素的索引。默认情况下，读写的值为数组从零开始指数.

*data_count*

[in] 值的数目应被读取。默认读取整个OpenCL缓冲区.

*mat*

© 2000-2025, MetaQuotes Ltd.

[out] 用于将数据写入缓冲区的矩阵可以是三种类型中的任何一种 — matrix、matrixf或matrixc.

*vec*

[out] 用于将数据写入缓冲区的向量可以是三种类型中的任何一种 — vector、vectorf或vectorc.

*rows=-1*

[in] 如果该参数被指定，也应该指定*cols*参数。如果没有指定新矩阵维度，则使用之前的矩阵维度。如果该值为-1，则行数不变.

*cols=-1*

[in] 如果该参数被指定，也应该跳过*rows*参数。矩阵遵循以下规则：要么两个参数都指定，要么都不指定，否则将发生错误。如果两个参数(*rows*和*cols*)都指定，则指定矩阵大小。-1情况下，则列数不变.

*size=-1*

[in] 如果没有指定该参数或参数值为-1，则向量长度不变.

返回值

读取元素个数。错误的情况下返回0。有关错误信息，使用 GetLastError() 函数.

如果矩阵或向量成功处理，返回true，否则返回false.

注意

对于一维数组，考虑到 AS_SERIES 旗帜，元素的数量，从写数据到一个OpenCL缓冲开始计算.

两个或两个以上的尺寸数组是一维的。在这种情况下，*data_offset* 是要在演示文稿中跳过的元素个数，而不是在第一维的元素个数.

使用方程式计算Pi的示例：

$$\pi = \lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N-1}\frac{4}{1+\left(\frac{2k+1}{2N}\right)^2} = 16\lim_{N\to\infty}N\sum_{k=0}^{N-1}\frac{1}{4N^2+(2k+1)^2}$$

```
#define  _num_steps         1000000000
#define  _divisor           40000
#define  _step              1.0 / _num_steps
#define  _intrnCnt          _num_steps / _divisor

//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
string D2S(double arg, int digits) { return DoubleToString(arg, digits); }
string I2S(int arg)                { return IntegerToString(arg); }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- OpenCL程序代码
const string clSource=
  "#define _step "+D2S(_step, 12)+"                    \r\n"
  "#define _intrnCnt "+I2S(_intrnCnt)+"                \r\n"
  "                                                    \r\n"
  "__kernel void Pi( __global double *out )            \r\n"
  "{                                                   \r\n"
  "  int i = get_global_id( 0 );                       \r\n"
  "  double partsum = 0.0;                             \r\n"
  "  double x = 0.0;                                   \r\n"
  "  long from = i * _intrnCnt;                        \r\n"
  "  long to = from + _intrnCnt;                       \r\n"
  "  for( long j = from; j < to; j ++ )                \r\n"
  "  {                                                 \r\n"
  "     x = ( j + 0.5 ) * _step;                       \r\n"
  "     partsum += 4.0 / ( 1. + x * x );               \r\n"
  "  }                                                 \r\n"
  "  out[ i ] = partsum;                               \r\n"
  "}                                                   \r\n";

//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart()
 {
  Print("Pi Calculation: step = "+D2S(_step, 12)+"; _intrnCnt = "+I2S(_intrnCnt));
//--- 比较OpenCL环境
  int clCtx;
  if((clCtx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
   {
    Print("OpenCL not found");
    return(-1);
   }
  int clPrg = CLProgramCreate(clCtx, clSource);
  int clKrn = CLKernelCreate(clPrg, "Pi");
  int clMem=CLBufferCreate(clCtx, _divisor*sizeof(double), CL_MEM_READ_WRITE);
  CLSetKernelArgMem(clKrn, 0, clMem);

  const uint offs[1]  = {0};
  const uint works[1] = {_divisor};
//--- 启动OpenCL程序
  ulong start=GetMicrosecondCount();
  if(!CLExecute(clKrn, 1, offs, works))
   {
    Print("CLExecute(clKrn, 1, offs, works) failed! Error ", GetLastError());
    CLFreeAll(clMem, clKrn, clPrg, clCtx);
    return(-1);
   }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 从OpenCL设备获取结果
  vector buffer(_divisor);
  if(!CLBufferRead(clMem, 0, buffer))
    {
     Print("CLBufferRead(clMem, 0, buffer) failed! Error ", GetLastError());
     CLFreeAll(clMem, clKrn, clPrg, clCtx);
     return(-1);
    }
//--- 将所有值加起来计算Pi
  double Pi=buffer.Sum()*_step;

  double time=(GetMicrosecondCount()-start)/1000.;
  Print("OpenCL: Pi calculated for "+D2S(time, 2)+" ms");
  Print("Pi = "+DoubleToString(Pi, 12));
//--- 空闲内存
  CLFreeAll(clMem, clKrn, clPrg, clCtx);
//--- 成功
  return(0);
 }
  /*
  Pi Calculation: step = 0.000000001000; _intrnCnt = 25000
  OpenCL: GPU device 'Ellesmere' selected
  OpenCL: Pi calculated for 99.98 ms
  Pi = 3.141592653590
  */
//+------------------------------------------------------------------+
//| 空闲内存的辅助程序                                                 |
//+------------------------------------------------------------------+
void CLFreeAll(const int clMem, const int clKrn, const int clPrg, const int clCtx)
 {
  CLBufferFree(clMem);
  CLKernelFree(clKrn);
  CLProgramFree(clPrg);
  CLContextFree(clCtx);
 }
```

© 2000-2025, MetaQuotes Ltd.

# CLExecute

此函数运行一个 OpenCL程序。有三个版本的函数：

1. 用一个内核启动内核函数

```
bool  CLExecute(
   int           kernel                  // 处理一个OpenCL程序的内核
   );
```

2. 启动几个内核副本（OpenCL函数）与任务空间描述

```
bool  CLExecute(
   int           kernel,                 // 处理一个OpenCL程序的内核
   uint          work_dim,               // 任务空间维度
   const uint&   global_work_offset[],   // 在任务空间初始偏移
   const uint&   global_work_size[]      // 任务总数
   );
```

3. 启动几个内核副本（OpenCL函数）与任务空间描述以及集合本地任务子集大小规范

```
bool  CLExecute(
   int           kernel,                 // 处理一个OpenCL程序的内核
   uint          work_dim,               // 任务空间维度
   const uint&   global_work_offset[],   // 在任务空间初始偏移
   const uint&   global_work_size[],     // 任务总数
   const uint&   local_work_size[]       // 在本地集合中的任务数
   );
```

参数

*kernel*
  [in] 处理OpenCL内核.

*work_dim*
  [in] 任务空间维度.

*global_work_offset[]*
  [in] 在任务空间初始偏移.

*global_work_size[]*
  [in] 任务子集大小.

*local_work_size[]*
  [in] 集合本地任务子集大小.

返回值

如果成功则返回true，否则返回false. 有关错误信息，使用 GetLastError() 函数.

注意

以下例子中要考虑参数的使用：

© 2000-2025, MetaQuotes Ltd.

- *work_dim* 指定 *work_items[]* 描述任务的阵列维度. 如果*work_dim*=3,三维阵列 *work_items[N1, N2, N3]* 将被使用.
- *global_work_size[]* 包含设定 *work_items[]*阵列大小的值. 如果 work_dim =3, *global_work_size[3]* 阵列将是 {40, 100, 320}. 然后我们将得到 *work_items[40, 100, 320]*. 所以,任务总数是 is 40 x 100 x 320 = 1 280 000.
- *local_work_size[]* 设定任务的子集将由OpenCL程序的指定内核来执行. 其维度等于 *work_items[]* 维度并允许共同任务子集精确分割成更小的子集. 事实上,应当选择 *local_work_size[]* 从而使全局任务设定分割成更小的子集. *local_work_size[3]={10, 10, 10}* 将适合当前的例子, 比如*work_items[40, 100, 320]* 能被 *local_items[10, 10, 10]* 阵列毫无剩余的收集起来.

© 2000-2025, MetaQuotes Ltd.

# CLExecutionStatus

返回OpenCL程序执行状态。

```
int  CLExecutionStatus(
   int   kernel            // 处理OpenCL程序的内核
   );
```

参数

*kernel*

[in] 处理OpenCL程序的内核。

返回值

返回OpenCL程序状态。该值可以是以下值的其中一个：

- CL_COMPLETE=0 -程序完成，
- CL_RUNNING=1 -运行，
- CL_SUBMITTED=2 -提交执行，
- CL_QUEUED=3 -排队，
- -1 (负1) -执行CLExecutionStatus()时出现错误。

© 2000-2025, MetaQuotes Ltd.

# 使用数据库

这些数据库函数使用了流行且易于使用的SQLite引擎。该引擎的便利之处在于，整个数据库都位于用户PC硬盘上的单个文件中。

这些函数可以方便地创建表格、向表中添加数据、执行更改和使用简单的SQL请求进行采样：

- 接收任何格式的交易历史和报价，
- 保存优化和测试结果，
- 通过其他分析组合准备和交换数据，
- 存储MQL5应用程序设置和状态。

查询允许使用统计和数学函数。

数据库函数可以使您通过SQL请求替换重复率最高的大数据组处理操作，因此常常可以使用DatabaseExecute/DatabasePrepare调用，而不是编写复杂的循环和比较。使用DatabaseReadBind函数，方便地获取现有架构中的查询结果。该函数运行在单个调用中一次性读取所有记录字段。

若要加快读取、编写和更改速度，可以使用DATABASE_OPEN_MEMORY标识在RAM中打开/创建数据库，虽然这样的数据库只适用于特定的应用程序，不可共享。当处理位于硬盘上的数据库时，应该使用DatabaseTransactionBegin/DatabaseTransactionCommit/DatabaseTransactionRollback将批量数据插入/更改封装在交易事务中。这可以使整个过程加快数百倍。

若要开始使用函数，请参阅文章SQLite：本地操作MQL5中的SQL数据库。

| 函数 | 操作 |
| --- | --- |
| DatabaseOpen | 在指定文件中打开或创建数据库 |
| DatabaseClose | 关闭数据库 |
| DatabaseImport | 从文件导入数据到表格中 |
| DatabaseExport | 将表格或SQL请求执行结果到处到CSV文件 |
| DatabasePrint | 在专家日志中打印表格或SQL执行结果 |
| DatabaseTableExists | 检查数据库中是否存在表格 |
| DatabaseExecute | 执行对指定数据库的请求 |
| DatabasePrepare | 创建可使用DatabaseRead()执行的请求句柄 |
| DatabaseReset | 重置请求，比如调用DatabasePrepare()之后 |
| DatabaseBind | 在请求中设置一个参数值 |
| DatabaseBindArray | 将数组设置为参数值 |
| DatabaseRead | 作为请求结果，移到下一个条目 |
| DatabaseFinalize | 移除在DatabasePrepare()中创建的请求 |
| DatabaseTransactionBegin | 开始事务执行 |
| DatabaseTransactionCommit | 完成事务执行 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 操作 |
| --- | --- |
| DatabaseTransactionRollback | 回滚事务 |
| DatabaseColumnsCount | 获取请求中的字段数 |
| DatabaseColumnName | 按索引获取字段名 |
| DatabaseColumnType | 按索引获取字段类型 |
| DatabaseColumnSize | 获取字段大小（以字节为单位） |
| DatabaseColumnText | 从当前记录中获取作为字符串的字段值 |
| DatabaseColumnInteger | 从当前记录中获取int类型的值 |
| DatabaseColumnLong | 从当前记录中获取long类型的值 |
| DatabaseColumnDouble | 从当前记录中获取double类型的值 |
| DatabaseColumnBlob | 从当前记录中获取作为数组的字段值 |

统计函数：

- mode - 模式
- median - median中值（50百分位）
- percentile_25 - 25百分位
- percentile_75
- percentile_90
- percentile_95
- percentile_99
- stddev或stddev_samp —采样标准偏差
- stddev_pop —总体标准偏差
- variance或var_samp —采样方差
- var_pop —总体方差

数学函数

- acos(X) -反余弦弧度值
- acosh(X) -双曲反余弦
- asin(X) -反正弦弧度值
- asinh(X) -双曲反正弦
- atan(X) -反正切弧度值
- atan2(X,Y) -X/Y比的反正切弧度值
- atanh(X) -双曲反正切
- ceil(X) -四舍五入到整数
- ceiling(X) -四舍五入到整数
- cos(X) -角余弦弧度值

© 2000-2025, MetaQuotes Ltd.

- cosh(X) -双曲余弦
- degrees(X) -将弧度转换为角度
- exp(X) -指数
- floor(X) -四舍五入到整数
- ln(X) -自然对数
- log(B,X) -指示基数的对数
- log(X) -小数对数
- log10(X) -小数对数
- log2(X) -基数2的对数
- mod(X,Y) -余数的除数
- pi() -近似Pi
- pow(X,Y) -指示基数的幂数
- power(X,Y) -指示基数的幂数
- radians(X) -将角度转换为弧度
- sin(X) -角正弦弧度值
- sinh(X) -双曲正弦
- sqrt(X) -平方根
- tan(X) -角正切弧度值
- tanh(X) -双曲正切
- trunc(X) -截取到最接近0的整数

例如：

```
select
  count(*) as book_count,
  cast(avg(parent) as integer) as mean,
  cast(median(parent) as integer) as median,
  mode(parent) as mode,
  percentile_90(parent) as p90,
  percentile_95(parent) as p95,
  percentile_99(parent) as p99
from moz_bookmarks;
```

© 2000-2025, MetaQuotes Ltd.

# DatabaseOpen

在指定文件中打开或创建数据库。

```
int  DatabaseOpen(
   string  filename,       // 文件名
   uint    flags           // 标识组合
   );
```

参数

*filename*

[in] 相对于"MQL5\Files"文件夹的文件名称。

*flags*

[in] ENUM_DATABASE_OPEN_FLAGS枚举中的标识组合。

返回值

如果执行成功，函数返回用于访问数据库的数据库句柄。否则，返回INVALID_HANDLE。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) -重要运行时错误；
- ERR_WRONG_INTERNAL_PARAMETER (4002) -访问"MQL5\Files"文件夹时出现的内部错误；
- ERR_INVALID_PARAMETER (4003) -数据库文件的路径包含空字符串，或设置不兼容的标识组合；
- ERR_NOT_ENOUGH_MEMORY (4004) -内存不足；
- ERR_WRONG_FILENAME (5002) -错误的数据库文件名称；
- ERR_TOO_LONG_FILENAME (5003) -数据库文件的绝对路径超过最大长度；
- ERR_DATABASE_TOO_MANY_OBJECTS (5122) -超过可接受的数据库对象的最大数目；
- ERR_DATABASE_CONNECT (5123) -数据库连接错误；
- ERR_DATABASE_MISUSE (5621) -错误使用SQLite库。

注意

如果*filename*参数为NULL或是空字符串""，则在磁盘上创建一个临时文件。并会在关闭数据库连接后自动删除。

如果*filename*参数为":memory:"，则数据库在内存中创建，并在与其连接关闭后自动删除。

如果*flags*参数没有 DATABASE_OPEN_READONLY或DATABASE_OPEN_READWRITE标识，则使用DATABASE_OPEN_READWRITE标识。

如果没有指定文件扩展名，请使用".sqlite"。

ENUM_DATABASE_OPEN_FLAGS

| ID | 描述 |
| --- | --- |
| DATABASE_OPEN_READONLY | 只读 |
| DATABASE_OPEN_READWRITE | 为读写开放 |
| DATABASE_OPEN_CREATE | 必要时，在磁盘上创建文件 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| DATABASE_OPEN_MEMORY | 在RAM中创建数据库 |
| DATABASE_OPEN_COMMON | 该文件位于所有程序端的通用文件夹 |

另见

DatabaseClose

© 2000-2025, MetaQuotes Ltd.

# DatabaseClose

关闭数据库。

```
void  DatabaseClose(
   int  database      // 在DatabaseOpen中接收的数据库句柄
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

返回值

无。

注意

调用DatabaseClose之后，对数据库的所有请求句柄都将被自动删除并变为无效。

如果句柄无效，函数设置ERR_DATABASE_INVALID_HANDLE错误。您可以使用GetLastError()检查错误。

另见

DatabaseOpen、DatabasePrepare

© 2000-2025, MetaQuotes Ltd.

# DatabaseImport

从文件导入数据到表格中。

```
long  DatabaseImport(
   int           database,          // 在DatabaseOpen中接收的数据库句柄
   const string  table,             // 要插入数据的表格名称
   const string  filename,          // 要导入数据的文件名
   uint          flags,             // 标识组合
   const string  separator,         // 数据分隔符
   ulong         skip_rows,         // 要跳过多少初始字符串
   const string  skip_comments      // 定义注释的字符串
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*table*

[in] 将要添加文件数据的表格名称。

*filename*

[in] 用于读取数据的CSV文件或ZIP存档文件。该名称可能包含子目录，并且是相对于MQL5\Files文件夹设置。

*flags*

[in] ENUM_DATABASE_IMPORT_FLAGS枚举中的标识组合。

*separator*

[in] CSV文件中的数据分隔符。

*skip_rows*

[in] 当从文件读取数据时，要跳过的初始字符串数。

*skip_comments*

[in] 用于将字符串指定为注释的字符串。如果在字符串的开头检测到*skip_comments*中的任何字符，那么该字符串将被视为注释，并不被导入。

返回值

返回导入字符串的数量，如果出现错误则返回-1。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) - 没有指定表格名称（空字符串或NULL）；
- ERR_DATABASE_INTERNAL (5120) - 内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) - 无效数据库句柄。

注意

如果没有名为*table*的表格，则自动生成该表。根据文件数据自动定义创建表格中的名称和字段类型。

© 2000-2025, MetaQuotes Ltd.

ENUM_DATABASE_IMPORT_FLAGS

| ID | 描述 |
| --- | --- |
| DATABASE_IMPORT_HEADER | 第一行包含表格字段的名称 |
| DATABASE_IMPORT_CRLF | CRLF（默认为LF）被认为是字符串中断 |
| DATABASE_IMPORT_APPEND | 将数据添加到现有表格末端 |
| DATABASE_IMPORT_QUOTED_STRINGS | 用双引号括住字符串值 |
| DATABASE_IMPORT_COMMON_FOLDER | 该文件存储在所有客户端的常规文件夹 \Terminal\Common\Files。 |

从DatabaseExport示例中的代码创建的文件中读取表格的示例：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
 {
  string csv_filename;
//--- 从客户端的通用文件夹获取要下载的文本文件的名称
  string filenames[];
  if(FileSelectDialog("Select a CSV file to download a table", NULL,
                      "Text files (*.csv)|*.csv",
                      FSD_WRITE_FILE|FSD_COMMON_FOLDER, filenames, "data.csv")>0)
   {
    //--- 显示每个选定文件的名称
    if(ArraySize(filenames)==1)
      csv_filename=filenames[0];
    else
     {
      Print("Unknown error while selecting file. Error code ", GetLastError());
      return;
     }
   }
  else
   {
    Print("CSV file not selected");
    return;
   }
//--- 创建或打开数据库
  string db_filename="test.sqlite";
  int db=DatabaseOpen(db_filename, DATABASE_OPEN_READWRITE|DATABASE_OPEN_CREATE);
//--- 检查TEST表格是否存在
  if(DatabaseTableExists(db, "TEST"))
   {
    //--- 移除TEST表格
```

© 2000-2025, MetaQuotes Ltd.

```
    if(!DatabaseExecute(db, "DROP TABLE IF EXISTS TEST"))
     {
      Print("Failed to drop the TEST table with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   }
  //--- 从文件导入条目到TEST表格
  long imported=DatabaseImport(db, "TEST", csv_filename, DATABASE_IMPORT_HEADER|DATABA
  if(imported>0)
   {
    Print(imported," lines imported in table TEST");
    DatabasePrint(db,"SELECT * FROM TEST",DATABASE_PRINT_NO_INDEX);
   }
  else
    {
     Print("DatabaseImport() failed. Error ",GetLastError());
    }
//--- 关闭数据库文件并告知
  DatabaseClose(db);
  PrintFormat("Database: %s closed", db_filename);
 }
```

另见

DatabaseOpen、DatabasePrint

© 2000-2025, MetaQuotes Ltd.

# DatabaseExport

将表格或SQL请求执行结果到处到CSV文件。该文件使用UTF-8编码创建。

```
long  DatabaseExport(
   int           database,          //在DatabaseOpen中接收的数据库句柄
   const string  table_or_sql,      // 表格名称或SQL请求
   const string  filename,          // 用于数据导出的CSV文件名称
   uint          flags,             // 标识组合
   const string  separator          // CSV文件中的数据分隔符
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*table_or_sql*

[in] 结果将被导出到指定文件的表格名称或SQL请求文本。

*filename*

[in] 用于数据导出的文件名。设置相对于MQL5\Files文件夹的路径。

*flags*

[in] ENUM_DATABASE_EXPORT_FLAGS枚举中的标识组合。

*separator*

[in] 数据分隔符。如果指定NULL，则将'\t'制表符用作分隔符。空字符串""被认为是有效分隔符，但不能将获取的CSV文件读取为表格 -它被认为是一组字符串。

返回值

返回导出条目的数量，或者在错误情况下返回负值。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) -重要运行时错误；
- ERR_INVALID_PARAMETER (4003) -数据库文件的路径包含空字符串，或设置不兼容的标识组合；
- ERR_NOT_ENOUGH_MEMORY (4004) -内存不足；
- ERR_FUNCTION_NOT_ALLOWED(4014) -不允许指定渠道；
- ERR_PROGRAM_STOPPED(4022) -操作取消（MQL程序停止）；
- ERR_WRONG_FILENAME (5002) -无效文件名；
- ERR_TOO_LONG_FILENAME (5003) -文件的绝对路径超过最大长度；
- ERR_CANNOT_OPEN_FILE(5004) -无法打开文件进行编写；
- ERR_FILE_WRITEERROR(5026) -无法写入文件；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_QUERY_PREPARE(5125) -请求生成错误；
- ERR_DATABASE_QUERY_NOT_READONLY -允许只读请求。

© 2000-2025, MetaQuotes Ltd.

注意

如果请求结果被导出，则SQL请求应该以"SELECT" 或 "select"开始。换句话说，SQL请求不能改变数据库状态，否则DatabaseExport()失败并显示错误。

数据库字符串值可能包含转换字符('\r'或'\r\n')，以及*分隔符*参数中设置的值分隔符。在这种情况下，请确保在'flags'参数中使用DATABASE_EXPORT_QUOTED_STRINGS标识。如果该标识已存在，则所有显示的字符串用双引号表示。如果一个字符串包含双引号，则用双层双引号代替。

ENUM_DATABASE_EXPORT_FLAGS

| ID | 描述 |
| --- | --- |
| DATABASE_EXPORT_HEADER | 在第一个字符串中显示字段名称 |
| DATABASE_EXPORT_INDEX | 显示字符串指数 |
| DATABASE_EXPORT_NO_BOM | 文件开头不要插入BOM掩码（默认插入BOM） |
| DATABASE_EXPORT_CRLF | 用CRLF进行字符串中断（默认为LF） |
| DATABASE_EXPORT_APPEND | 将数据添加到现有文件的末尾（默认情况下，文件被覆盖）。如果文件不存在，则将被创建。 |
| DATABASE_EXPORT_QUOTED_STRINGS | 用双引号显示字符串值。 |
| DATABASE_EXPORT_COMMON_FOLDER | CSV文件在所有客户端的常规文件夹\Terminal\Common\File创建。 |

例如：

```
input int InpRates=100;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
 {
  MqlRates rates[];
//--- 记住收到柱形图的开始时间
  ulong start=GetMicrosecondCount();
//--- 在H1上请求最后100柱形图
  if(CopyRates(Symbol(), PERIOD_H1, 1, InpRates, rates)<InpRates)
   {
    Print("CopyRates() failed,, Error ", GetLastError());
    return;
   }
  else
```

© 2000-2025, MetaQuotes Ltd.

```
   {
    //--- 接收多少柱形图，以及接收所花费的时间
    PrintFormat("%s: CopyRates received %d bars in %d ms ",
                _Symbol, ArraySize(rates), (GetMicrosecondCount()-start)/1000);
   }
//--- 设置存储数据库的文件名称
  string filename=_Symbol+"_"+EnumToString(PERIOD_H1)+"_"+TimeToString(TimeCurrent())+
  StringReplace(filename, ":", "-"); // 文件名中不允许使用":"字符
//--- 在常规程序端文件夹中打开/创建数据库
  int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE|DATABASE_OPEN_CREATE|DATABASE_
  if(db==INVALID_HANDLE)
   {
    Print("Database: ", filename, " open failed with code ", GetLastError());
    return;
   }
  else
    Print("Database: ", filename, " opened successfully");

//--- 检查RATES表格是否存在
  if(DatabaseTableExists(db, "RATES"))
   {
    //--- 移除RATES表格
    if(!DatabaseExecute(db, "DROP TABLE IF EXISTS RATES"))
     {
      Print("Failed to drop the RATES table with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   }
//--- 创建RATES表格
  if(!DatabaseExecute(db, "CREATE TABLE RATES("
                      "SYMBOL             CHAR(10),"
                      "TIME               INT NOT NULL,"
                      "OPEN               REAL,"
                      "HIGH               REAL,"
                      "LOW                REAL,"
                      "CLOSE              REAL,"
                      "TICK_VOLUME        INT,"
                      "SPREAD             INT,"
                      "REAL_VOLUME        INT);"))
   {
    Print("DB: ", filename, " create table RATES with code ", GetLastError());
    DatabaseClose(db);
    return;
   }
//--- 显示RATES表格中所有字段列表
  if(DatabasePrint(db, "PRAGMA TABLE_INFO(RATES)", 0)<0)
   {
    PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(RATES)\") failed, error code=%d at
```

© 2000-2025, MetaQuotes Ltd.

```
     DatabaseClose(db);
     return;
    }
//--- 创建参数化请求，将柱形图添加到RATES表格中
  string sql="INSERT INTO RATES (SYMBOL,TIME,OPEN,HIGH,LOW,CLOSE,TICK_VOLUME,SPREAD,RE
             " VALUES (?1,?2,?3,?4,?5,?6,?7,?8,?9)"; // 请求参数
  int request=DatabasePrepare(db, sql);
  if(request==INVALID_HANDLE)
    {
     PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
     Print("SQL request: ", sql);
     DatabaseClose(db);
     return;
    }
//--- 设置第一个请求参数的值
  DatabaseBind(request, 0, _Symbol);
//--- 记住将柱形图添加到RATES表格的开始时间
  start=GetMicrosecondCount();
  DatabaseTransactionBegin(db);
  int total=ArraySize(rates);
  bool request_error=false;
  for(int i=0; i<total; i++)
    {
     //--- 在添加条目之前设置其余参数的值
     ResetLastError();
     if(!DatabaseBind(request, 1, rates[i].time))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
     //--- 如果之前的DatabaseBind()调用成功，则设置下一个参数
     if(!request_error && !DatabaseBind(request, 2, rates[i].open))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
     if(!request_error && !DatabaseBind(request, 3, rates[i].high))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
     if(!request_error && !DatabaseBind(request, 4, rates[i].low))
       {
```

© 2000-2025, MetaQuotes Ltd.

```
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    if(!request_error && !DatabaseBind(request, 5, rates[i].close))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    if(!request_error && !DatabaseBind(request, 6, rates[i].tick_volume))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    if(!request_error && !DatabaseBind(request, 7, rates[i].spread))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    if(!request_error && !DatabaseBind(request, 8, rates[i].real_volume))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }

    //--- 执行插入条目的请求并检查错误
    if(!request_error && !DatabaseRead(request) && (GetLastError()!=ERR_DATABASE_NO_MO
     {
      PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
      DatabaseFinalize(request);
      request_error=true;
      break;
     }
    //--- 在下一次参数更新之前重置请求
    if(!request_error && !DatabaseReset(request))
     {
      PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
      DatabaseFinalize(request);
      request_error=true;
      break;
```

© 2000-2025, MetaQuotes Ltd.

```
      }
    } //--- 完成检查所有柱形图

//--- 交易事务状态
  if(request_error)
    {
     PrintFormat("Table RATES: failed to add %d bars ", ArraySize(rates));
     DatabaseTransactionRollback(db);
     DatabaseClose(db);
     return;
    }
  else
    {
     DatabaseTransactionCommit(db);
     PrintFormat("Table RATES: added %d bars in %d ms",
                 ArraySize(rates), (GetMicrosecondCount()-start)/1000);
    }
//--- 将RATES表格保存为CSV文件
  string csv_filename=Symbol()+".csv";
  long saved=DatabaseExport(db, "SELECT * FROM RATES", csv_filename, DATABASE_EXPORT_H
  if(saved>0)
    Print("Table RATES saved in ", Symbol(), ".csv");
  else
    Print("DatabaseExport() failed. Error ", GetLastError());
//--- 关闭数据库文件并告知
  DatabaseClose(db);
  PrintFormat("Database: %s created and closed", filename);
```

另见

DatabasePrint、DatabaseImport

© 2000-2025, MetaQuotes Ltd.

# DatabasePrint

在专家日志中打印表格或SQL执行结果。

```
long  DatabasePrint(
   int           database,          // 在DatabaseOpen中接收的数据库句柄
   const string  table_or_sql,      // 表格或SQL请求
   uint          flags              // 标识组合
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*table_or_sql*

[in] 结果显示在专家日志中的表格名称或SQL请求文本。

*flags*

[in] 轻易输出格式的标识组合。标识定义如下：

DATABASE_PRINT_NO_HEADER -不显示表格的列名（字段名）
DATABASE_PRINT_NO_INDEX -不显示字符串索引
DATABASE_PRINT_NO_FRAME -不显示分隔标题和数据的框架
DATABASE_PRINT_STRINGS_RIGHT -将字符串向右对齐。

如果flags=0，则显示列和字符串，标题和数据用框架分隔，而字符串向左对齐。

返回值

返回导出字符串的数量，如果出现错误则返回-1。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) -重要运行时错误；
- ERR_NOT_ENOUGH_MEMORY (4004) -内存不足；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；

注意

如果日志显示请求结果，则SQL请求应该以"SELECT" 或 "select"开始。换句话说，SQL请求不能改变数据库状态，否则DatabasePrint()失败并显示错误。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   string filename="departments.sqlite";
//--- 在常规程序端文件夹中创建或打开数据库
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE |DATAB
   if(db==INVALID_HANDLE)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       Print("DB: ", filename, " open failed with code ", GetLastError());
       return;
      }

//--- 创建COMPANY表格
   if(!CreataTableCompany(db))
      {
       DatabaseClose(db);
       return;
      }
//--- 创建DEPARTMENT表格
   if(!CreataTableDepartment(db))
      {
       DatabaseClose(db);
       return;
      }

//--- 显示COMPANY和DEPARTMENT表格中所有字段的列表
   PrintFormat("Try to print request \"PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(DE
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(DEPARTMENT)", 0)
      {
       PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO()\") failed, error code=%d", GetL
       DatabaseClose(db);
       return;
      }
//--- 在日志中显示COMPANY表格
   PrintFormat("Try to print request \"SELECT * from COMPANY\"");
   if(DatabasePrint(db, "SELECT * from COMPANY", 0)<0)
      {
       Print("DatabasePrint failed with code ", GetLastError());
       DatabaseClose(db);
       return;
      }
//--- 请求合并COMPANY和DEPARTMENT表格的文本
   string request="SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT "
                  "ON COMPANY.ID = DEPARTMENT.EMP_ID";
//--- 显示表格合并结果
   PrintFormat("Try to print request \"SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUT
   if(DatabasePrint(db, request, 0)<0)
      {
       Print("DatabasePrint failed with code ", GetLastError());
       DatabaseClose(db);
       return;
      }
//--- 关闭数据库
   DatabaseClose(db);
  }
/*
```

© 2000-2025, MetaQuotes Ltd.

```
结论：
Try to print request "PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(DEPARTMENT)"
#| cid name    type     notnull dflt_value pk
-+-----------------------------------------
1|   0 ID      INT            1             1
2|   1 NAME    TEXT           1             0
3|   2 AGE     INT            1             0
4|   3 ADDRESS CHAR(50)       0             0
5|   4 SALARY  REAL           0             0
#| cid name   type     notnull dflt_value pk
-+-----------------------------------------
1|   0 ID     INT            1             1
2|   1 DEPT   CHAR(50)       1             0
3|   2 EMP_ID INT            1             0
Try to print request "SELECT * from COMPANY"
#| ID NAME  AGE ADDRESS     SALARY
-+--------------------------------
1|  1 Paul   32 California 25000.0
2|  2 Allen  25 Texas      15000.0
3|  3 Teddy  23 Norway     20000.0
4|  4 Mark   25 Rich-Mond  65000.0
5|  5 David  27 Texas      85000.0
6|  6 Kim    22 South-Hall 45000.0
7|  7 James  24 Houston    10000.0
Try to print request "SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMEN
#| EMP_ID NAME  DEPT
-+-------------------------
1|      1 Paul  IT Billing
2|      2 Allen Engineering
3|        Teddy
4|        Mark
5|        David
6|        Kim
7|      7 James Finance
*/
//+------------------------------------------------------------------+
//| 创建COMPANY表格                                                  |
//+------------------------------------------------------------------+
bool CreateTableCompany(int database)
  {
//--- 如果COMPANY表格已存在，请将其删除
   if(DatabaseTableExists(database, "COMPANY"))
     {
      //--- 删除表格
      if(!DatabaseExecute(database, "DROP TABLE COMPANY"))
        {
         Print("Failed to drop table COMPANY with code ", GetLastError());
         return(false);
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 创建COMPANY表格
   if(!DatabaseExecute(database, "CREATE TABLE COMPANY("
                       "ID INT PRIMARY KEY     NOT NULL,"
                       "NAME           TEXT    NOT NULL,"
                       "AGE            INT     NOT NULL,"
                       "ADDRESS        CHAR(50),"
                       "SALARY         REAL );"))
      {
       Print("DB: create table COMPANY failed with code ", GetLastError());
       return(false);
      }

//--- 将数据输入到COMPANY表格
   if(!DatabaseExecute(database, "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VAI
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'A
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, 'T
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, 'M
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (5, 'I
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (6, 'P
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (7, 'J
      {
       Print("COMPANY insert failed with code ", GetLastError());
       return(false);
      }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
//| 创建DEPARTMENT表格                                               |
//+------------------------------------------------------------------+
bool CreateTableDepartment(int database)
  {
//--- 如果DEPARTMENT表格已存在，请将其删除
   if(DatabaseTableExists(database, "DEPARTMENT"))
      {
       //--- 删除表格
       if(!DatabaseExecute(database, "DROP TABLE DEPARTMENT"))
         {
          Print("Failed to drop table DEPARTMENT  with code ", GetLastError());
          return(false);
         }
      }
//--- 创建DEPARTMENT表格
   if(!DatabaseExecute(database, "CREATE TABLE DEPARTMENT ("
                       "ID      INT PRIMARY KEY   NOT NULL,"
                       "DEPT    CHAR(50)          NOT NULL,"
                       "EMP_ID  INT               NOT NULL);"))
      {
```

© 2000-2025, MetaQuotes Ltd.

```
      Print("DB: create table DEPARTMENT failed with code ", GetLastError());
      return(false);
     }

//--- 将数据输入到DEPARTMENT表格
   if(!DatabaseExecute(database, "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (1, '
                       "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (2, 'Engineerin
                       "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (3, 'Finance',
     {
      Print("DEPARTMENT insert failed with code ", GetLastError());
      return(false);
     }
//--- 成功
   return(true);
  }
//+------------------------------------------------------------------+
```

另见

DatabaseExport、DatabaseImport

© 2000-2025, MetaQuotes Ltd.

# DatabaseTableExists

检查数据库中是否存在表格。

```
bool  DatabaseTableExists(
   int     database,      // 在DatabaseOpen中接收的数据库句柄
   string  table          // 表格名称
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*table*

[in] 表格名称。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) - 没有指定表格名称（空字符串或NULL）；
- ERR_WRONG_STRING_PARAMETER (5040) - 将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) - 内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) - 无效数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误；
- ERR_DATABASE_NO_MORE_DATA (5126) - 不存在表格（没有错误，正常完成）。

另见

DatabasePrepare、DatabaseFinalize

© 2000-2025, MetaQuotes Ltd.

# DatabaseExecute

执行对指定数据库的请求。

```
bool  DatabaseExecute(
   int     database,       // 在DatabaseOpen中接收的数据库句柄
   string  sql             // SQL请求
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*sql*

[in] SQL request.

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) - 重要运行时错误；
- ERR_INVALID_PARAMETER (4003) - sql参数包含空字符串；
- ERR_NOT_ENOUGH_MEMORY (4004) - 内存不足；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误。

**Example:**

```
//--- symbol statistics
struct Symbol_Stats
  {
   string            name;             // symbol name
   int               trades;           // number of trades for the symbol
   double            gross_profit;     // total profit for the symbol
   double            gross_loss;       // total loss for the symbol
   double            total_commission; // total commission for the symbol
   double            total_swap;       // total swaps for the symbol
   double            total_profit;     // total profit excluding swaps and commissions
   double            net_profit;       // net profit taking into account swaps and co
   int               win_trades;       // number of profitable trades
   int               loss_trades;      // number of losing trades
   double            expected_payoff;  // expected payoff for the trade excluding swa
   double            win_percent;      // percentage of winning trades
   double            loss_percent;     // percentage of losing trades
   double            average_profit;   // average profit
   double            average_loss;     // average loss
   double            profit_factor;    // profit factor
  };
```

© 2000-2025, MetaQuotes Ltd.

```
//--- Magic Number statistics
struct Magic_Stats
  {
   long              magic;            // EA's Magic Number
   int               trades;           // number of trades for the symbol
   double            gross_profit;     // total profit for the symbol
   double            gross_loss;       // total loss for the symbol
   double            total_commission; // total commission for the symbol
   double            total_swap;       // total swaps for the symbol
   double            total_profit;     // total profit excluding swaps and commissions
   double            net_profit;       // net profit taking into account swaps and com
   int               win_trades;       // number of profitable trades
   int               loss_trades;      // number of losing trades
   double            expected_payoff;  // expected payoff for the trade excluding swap
   double            win_percent;      // percentage of winning trades
   double            loss_percent;     // percentage of losing trades
   double            average_profit;   // average profit
   double            average_loss;     // average loss
   double            profit_factor;    // profit factor
  };

//--- entry hour statistics
struct Hour_Stats
  {
   char              hour_in;          // market entry hour
   int               trades;           // number of trades in this entry hour
   double            volume;           // volume of trades in this entry hour
   double            gross_profit;     // total profit in this entry hour
   double            gross_loss;       // total loss in this entry hour
   double            net_profit;       // net profit taking into account swaps and com
   int               win_trades;       // number of profitable trades
   int               loss_trades;      // number of losing trades
   double            expected_payoff;  // expected payoff for the trade excluding swap
   double            win_percent;      // percentage of winning trades
   double            loss_percent;     // percentage of losing trades
   double            average_profit;   // average profit
   double            average_loss;     // average loss
   double            profit_factor;    // profit factor
  };

int ExtDealsTotal=0;;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
   string filename=IntegerToString(AccountInfoInteger(ACCOUNT_LOGIN))+"_stats.sqlite";
```

© 2000-2025, MetaQuotes Ltd.

```
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- create the DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }
   PrintFormat("Deals in the trading history: %d ", ExtDealsTotal);

//--- get trading statistics per symbols
   int request=DatabasePrepare(db, "SELECT r.*,"
                               "   (case when r.trades != 0 then (r.gross_profit+r.gro
                               "   (case when r.trades != 0 then r.win_trades*100.0/r.
                               "   (case when r.trades != 0 then r.loss_trades*100.0/r
                               "   r.gross_profit/r.win_trades as average_profit,"
                               "   r.gross_loss/r.loss_trades as average_loss,"
                               "   (case when r.gross_loss!=0.0 then r.gross_profit/(-
                               "FROM "
                               "   ("
                               "   SELECT SYMBOL,"
                               "   sum(case when entry =1 then 1 else 0 end) as trades
                               "   sum(case when profit > 0 then profit else 0 end) as
                               "   sum(case when profit < 0 then profit else 0 end) as
                               "   sum(swap) as total_swap,"
                               "   sum(commission) as total_commission,"
                               "   sum(profit) as total_profit,"
                               "   sum(profit+swap+commission) as net_profit,"
                               "   sum(case when profit > 0 then 1 else 0 end) as win_
                               "   sum(case when profit < 0 then 1 else 0 end) as loss
                               "   FROM DEALS "
                               "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                               "   GROUP BY SYMBOL"
                               "   ) as r");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Symbol_Stats stats[], symbol_stats;
   ArrayResize(stats, ExtDealsTotal);
   int i=0;
//--- get records from request results
```

© 2000-2025, MetaQuotes Ltd.

```
   for(; DatabaseReadBind(request, symbol_stats) ; i++)
     {
      stats[i].name=symbol_stats.name;
      stats[i].trades=symbol_stats.trades;
      stats[i].gross_profit=symbol_stats.gross_profit;
      stats[i].gross_loss=symbol_stats.gross_loss;
      stats[i].total_commission=symbol_stats.total_commission;
      stats[i].total_swap=symbol_stats.total_swap;
      stats[i].total_profit=symbol_stats.total_profit;
      stats[i].net_profit=symbol_stats.net_profit;
      stats[i].win_trades=symbol_stats.win_trades;
      stats[i].loss_trades=symbol_stats.loss_trades;
      stats[i].expected_payoff=symbol_stats.expected_payoff;
      stats[i].win_percent=symbol_stats.win_percent;
      stats[i].loss_percent=symbol_stats.loss_percent;
      stats[i].average_profit=symbol_stats.average_profit;
      stats[i].average_loss=symbol_stats.average_loss;
      stats[i].profit_factor=symbol_stats.profit_factor;
     }
   ArrayResize(stats, i);
   Print("Trade statistics by Symbol");
   ArrayPrint(stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- get trading statistics for Expert Advisors by Magic Numbers
   request=DatabasePrepare(db, "SELECT r.*,"
                           "   (case when r.trades != 0 then (r.gross_profit+r.gross_l
                           "   (case when r.trades != 0 then r.win_trades*100.0/r.trad
                           "   (case when r.trades != 0 then r.loss_trades*100.0/r.tra
                           "   r.gross_profit/r.win_trades as average_profit,"
                           "   r.gross_loss/r.loss_trades as average_loss,"
                           "   (case when r.gross_loss!=0.0 then r.gross_profit/(-r.gr
                           "FROM "
                           "   ("
                           "   SELECT MAGIC,"
                           "   sum(case when entry =1 then 1 else 0 end) as trades,"
                           "   sum(case when profit > 0 then profit else 0 end) as gro
                           "   sum(case when profit < 0 then profit else 0 end) as gro
                           "   sum(swap) as total_swap,"
                           "   sum(commission) as total_commission,"
                           "   sum(profit) as total_profit,"
                           "   sum(profit+swap+commission) as net_profit,"
                           "   sum(case when profit > 0 then 1 else 0 end) as win_trad
                           "   sum(case when profit < 0 then 1 else 0 end) as loss_tra
                           "   FROM DEALS "
                           "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                           "   GROUP BY MAGIC"
```

© 2000-2025, MetaQuotes Ltd.

```
                              "   ) as r");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Magic_Stats EA_stats[], magic_stats;
   ArrayResize(EA_stats, ExtDealsTotal);
   i=0;
//--- print
   for(; DatabaseReadBind(request, magic_stats) ; i++)
     {
      EA_stats[i].magic=magic_stats.magic;
      EA_stats[i].trades=magic_stats.trades;
      EA_stats[i].gross_profit=magic_stats.gross_profit;
      EA_stats[i].gross_loss=magic_stats.gross_loss;
      EA_stats[i].total_commission=magic_stats.total_commission;
      EA_stats[i].total_swap=magic_stats.total_swap;
      EA_stats[i].total_profit=magic_stats.total_profit;
      EA_stats[i].net_profit=magic_stats.net_profit;
      EA_stats[i].win_trades=magic_stats.win_trades;
      EA_stats[i].loss_trades=magic_stats.loss_trades;
      EA_stats[i].expected_payoff=magic_stats.expected_payoff;
      EA_stats[i].win_percent=magic_stats.win_percent;
      EA_stats[i].loss_percent=magic_stats.loss_percent;
      EA_stats[i].average_profit=magic_stats.average_profit;
      EA_stats[i].average_loss=magic_stats.average_loss;
      EA_stats[i].profit_factor=magic_stats.profit_factor;
     }
   ArrayResize(EA_stats, i);
   Print("Trade statistics by Magic Number");
   ArrayPrint(EA_stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- make sure that hedging system for open position management is used on the accou
   if((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)!=ACCOUNT_MARGI
     {
      //--- deals cannot be transformed to trades using a simple method through transa
      DatabaseClose(db);
      return;
     }

//--- now create the TRADES table based on the DEALS table
   if(!CreateTableTrades(db))
     {
      DatabaseClose(db);
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
//--- fill in the TRADES table using an SQL query based on DEALS table data
   if(DatabaseTableExists(db, "DEALS"))
      //--- populate the TRADES table
      if(!DatabaseExecute(db, "INSERT INTO TRADES(TIME_IN,HOUR_IN,TICKET,TYPE,VOLUME,S
                          "SELECT "
                          "   d1.time as time_in,"
                          "   d1.hour as hour_in,"
                          "   d1.position_id as ticket,"
                          "   d1.type as type,"
                          "   d1.volume as volume,"
                          "   d1.symbol as symbol,"
                          "   d1.price as price_in,"
                          "   d2.time as time_out,"
                          "   d2.price as price_out,"
                          "   d1.commission+d2.commission as commission,"
                          "   d2.swap as swap,"
                          "   d2.profit as profit "
                          "FROM DEALS d1 "
                          "INNER JOIN DEALS d2 ON d1.position_id=d2.position_id "
                          "WHERE d1.entry=0 AND d2.entry=1     "))
        {
         Print("DB: fillng the table TRADES failed with code ", GetLastError());
         return;
        }

//--- get trading statistics by market entry hours
   request=DatabasePrepare(db, "SELECT r.*,"
                           "   (case when r.trades != 0 then (r.gross_profit+r.gross_l
                           "   (case when r.trades != 0 then r.win_trades*100.0/r.trad
                           "   (case when r.trades != 0 then r.loss_trades*100.0/r.tra
                           "   r.gross_profit/r.win_trades as average_profit,"
                           "   r.gross_loss/r.loss_trades as average_loss,"
                           "   (case when r.gross_loss!=0.0 then r.gross_profit/(-r.gr
                           "FROM "
                           "   ("
                           "   SELECT HOUR_IN,"
                           "   count() as trades,"
                           "   sum(volume) as volume,"
                           "   sum(case when profit > 0 then profit else 0 end) as gro
                           "   sum(case when profit < 0 then profit else 0 end) as gro
                           "   sum(profit) as net_profit,"
                           "   sum(case when profit > 0 then 1 else 0 end) as win_trad
                           "   sum(case when profit < 0 then 1 else 0 end) as loss_tra
                           "   FROM TRADES "
                           "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                           "   GROUP BY HOUR_IN"
                           "   ) as r");
```

© 2000-2025, MetaQuotes Ltd.

```
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Hour_Stats hours_stats[], h_stats;
   ArrayResize(hours_stats, ExtDealsTotal);
   i=0;
//--- print
   for(; DatabaseReadBind(request, h_stats) ; i++)
     {
      hours_stats[i].hour_in=h_stats.hour_in;
      hours_stats[i].trades=h_stats.trades;
      hours_stats[i].volume=h_stats.volume;
      hours_stats[i].gross_profit=h_stats.gross_profit;
      hours_stats[i].gross_loss=h_stats.gross_loss;
      hours_stats[i].net_profit=h_stats.net_profit;
      hours_stats[i].win_trades=h_stats.win_trades;
      hours_stats[i].loss_trades=h_stats.loss_trades;
      hours_stats[i].expected_payoff=h_stats.expected_payoff;
      hours_stats[i].win_percent=h_stats.win_percent;
      hours_stats[i].loss_percent=h_stats.loss_percent;
      hours_stats[i].average_profit=h_stats.average_profit;
      hours_stats[i].average_loss=h_stats.average_loss;
      hours_stats[i].profit_factor=h_stats.profit_factor;
     }
   ArrayResize(hours_stats, i);
   Print("Trade statistics by entry hour");
   ArrayPrint(hours_stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- close database
   DatabaseClose(db);
   return;
  }
/*
Deals in the trading history: 2771
Trade statistics by Symbol
      [name] [trades] [gross_profit] [gross_loss] [total_commission] [total_swap] [tot
[0] "AUDUSD"      112      503.20000   -568.00000           -8.83000    -24.64000
[1] "EURCHF"      125      607.71000   -956.85000          -11.77000    -45.02000
[2] "EURJPY"      127     1078.49000  -1057.83000          -10.61000    -45.76000
[3] "EURUSD"      233     1685.60000  -1386.80000          -41.00000    -83.76000
[4] "GBPCHF"      125     1881.37000  -1424.72000          -22.60000    -51.56000
[5] "GBPJPY"      127     1943.43000  -1776.67000          -18.84000    -52.46000
[6] "GBPUSD"      121     1668.50000  -1438.20000           -7.96000    -49.93000
```

© 2000-2025, MetaQuotes Ltd.

```
[7] "USDCAD"        99      405.28000   -475.47000           -8.68000    -31.68000
[8] "USDCHF"       206     1588.32000  -1241.83000          -17.98000    -65.92000
[9] "USDJPY"       107      464.73000   -730.64000          -35.12000    -34.24000

Trade statistics by Magic Number
    [magic] [trades] [gross_profit] [gross_loss] [total_commission] [total_swap] [tota
[0]     100      242     2584.80000  -2110.00000          -33.36000    -93.53000
[1]     200      254     3021.92000  -2834.50000          -29.45000    -98.22000
[2]     300      250     2489.08000  -2381.57000          -34.37000    -96.58000
[3]     400      224     1272.50000  -1283.00000          -24.43000    -64.80000
[4]     500      198     1141.23000  -1051.91000          -27.66000    -63.36000
[5]     600      214     1317.10000  -1396.03000          -34.12000    -68.48000

Trade statistics by entry hour
     [hour_in] [trades] [volume] [gross_profit] [gross_loss] [net_profit] [win_trades]
[ 0]         0       50  5.00000      336.51000   -747.47000   -410.96000           21
[ 1]         1       20  2.00000      102.56000    -57.20000     45.36000           12
[ 2]         2        6  0.60000       38.55000    -14.60000     23.95000            5
[ 3]         3       38  3.80000      173.84000   -200.15000    -26.31000           22
[ 4]         4       60  6.00000      361.44000   -389.40000    -27.96000           27
[ 5]         5       32  3.20000      157.43000   -179.89000    -22.46000           20
[ 6]         6       18  1.80000       95.59000   -162.33000    -66.74000           11
[ 7]         7       14  1.40000       38.48000   -134.30000    -95.82000            5
[ 8]         8       42  4.20000      368.48000   -322.30000     46.18000           24
[ 9]         9      118 11.80000     1121.62000   -875.21000    246.41000           72
[10]        10      206 20.60000     2280.59000  -2021.80000    258.79000          115
[11]        11      138 13.80000     1377.02000   -994.18000    382.84000           84
[12]        12      152 15.20000     1247.56000  -1463.80000   -216.24000           84
[13]        13       64  6.40000      778.27000   -516.22000    262.05000           36
[14]        14       62  6.20000      536.93000   -427.47000    109.46000           38
[15]        15       50  5.00000      699.92000   -413.00000    286.92000           28
[16]        16       88  8.80000      778.55000   -514.00000    264.55000           51
[17]        17       76  7.60000      533.92000  -1019.46000   -485.54000           44
[18]        18       52  5.20000      237.17000   -246.78000     -9.61000           24
[19]        19       52  5.20000      407.67000   -150.36000    257.31000           30
[20]        20       18  1.80000       65.92000    -89.09000    -23.17000            9
[21]        21       10  1.00000       41.86000    -32.38000      9.48000            7
[22]        22       14  1.40000       45.55000    -83.72000    -38.17000            6
[23]        23        2  0.20000        1.20000     -1.90000     -0.70000            1
*/

//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(database, "DEALS"))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID           INT KEY NOT NULL,"
                          "ORDER_ID     INT     NOT NULL,"
                          "POSITION_ID  INT     NOT NULL,"
                          "TIME         INT     NOT NULL,"
                          "TYPE         INT     NOT NULL,"
                          "ENTRY        INT     NOT NULL,"
                          "SYMBOL       CHAR(10),"
                          "VOLUME       REAL,"
                          "PRICE        REAL,"
                          "PROFIT       REAL,"
                          "SWAP         REAL,"
                          "COMMISSION   REAL,"
                          "MAGIC        INT,"
                          "HOUR         INT,"
                          "REASON       INT);"))
        {
         Print("DB: create the DEALS table  failed with code ", GetLastError());
         return(false);
        }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   ExtDealsTotal=HistoryDealsTotal();
//--- add deals to the table
   if(!InsertDeals(database))
      return(false);
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database        |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
     {
      Print("Failed to drop the DEALS table with code ", GetLastError());
      return(false);
     }
//--- the table has been successfully deleted
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database)
  {
//--- Auxiliary variables
   ulong    deal_ticket;          // deal ticket
   long     order_ticket;         // the ticket of the order by which the deal was exec
   long     position_ticket;      // ID of the position to which the deal belongs
   datetime time;                 // deal execution time
   long     type ;                // deal type
   long     entry ;               // deal direction
   string   symbol;               // the symbol fro which the deal was executed
   double   volume;               // operation volume
   double   price;                // price
   double   profit;               // financial result
   double   swap;                 // swap
   double   commission;           // commission
   long     magic;                // Magic number (Expert Advisor ID)
   long     reason;               // deal execution reason or source
   char     hour;                 // deal execution hour
   MqlDateTime time_strusture;
//--- go through all deals and add them to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
// --- lock the database before executing transactions
   DatabaseTransactionBegin(database);
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      TimeToStruct(time, time_strusture);
      hour= (char)time_strusture.hour;
      //--- add each deal to the table using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TIM
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G,
```

© 2000-2025, MetaQuotes Ltd.

```
                                        deal_ticket, order_ticket, position_ticket, tir
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%d with code %d", __FUNCTION__, deal_
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
         break;
        }
      }
//--- check for transaction execution errors
   if(failed)
     {
      //--- roll back all transactions and unlock the database
      DatabaseTransactionRollback(database);
      PrintFormat("%s: DatabaseExecute() failed with code ", __FUNCTION__, GetLastErro
      return(false);
     }
//--- all transactions have been performed successfully - record changes and unlock th
   DatabaseTransactionCommit(database);
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the TRADES table                                         |
//+------------------------------------------------------------------+
bool CreateTableTrades(int database)
  {
//--- if the TRADES table already exists, delete it
   if(!DeleteTable(database, "TRADES"))
      return(false);
//--- check if the table exists
   if(!DatabaseTableExists(database, "TRADES"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE TRADES("
                          "TIME_IN     INT     NOT NULL,"
                          "HOUR_IN     INT     NOT NULL,"
                          "TICKET      INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "VOLUME      REAL,"
                          "SYMBOL      CHAR(10),"
                          "PRICE_IN    REAL,"
                          "TIME_OUT    INT     NOT NULL,"
                          "PRICE_OUT   REAL,"
                          "COMMISSION  REAL,"
                          "SWAP        REAL,"
                          "PROFIT      REAL);"))
        {
         Print("DB: create the TRADES table failed with code ", GetLastError());
         return(false);
        }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
```

另 见

DatabasePrepare、DatabaseFinalize

© 2000-2025, MetaQuotes Ltd.

# DatabasePrepare

创建可使用DatabaseRead()执行的请求句柄。

```
int  DatabasePrepare(
   int      database,       // 在DatabaseOpen中接收的数据库句柄
   string   sql,            // SQL 请求
            ...             // 请求参数
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

*sql*

[in] SQL请求，可能包含自动替换参数，其名为?1,?2,...

*...*

[in] 自动替换请求参数。

返回值

如果成功，函数返回SQL请求句柄。否则，返回INVALID_HANDLE。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) -数据库文件的路径包含空字符串，或设置不兼容的标识组合；
- ERR_NOT_ENOUGH_MEMORY (4004) -内存不足；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_TOO_MANY_OBJECTS (5122) -超过可接受的数据库对象的最大数目；
- ERR_DATABASE_PREPARE (5125) -请求生成错误。

注意

DatabasePrepare()函数不执行对数据库的请求。其目的是验证请求参数，并根据验证结果返回执行SQL请求的句柄。请求本身在第一次调用DatabaseRead()期间进行设置。

**Example:**

```
//--- Structure to store the deal
struct Deal
  {
   ulong             ticket;           // DEAL_TICKET
   long              order_ticket;     // DEAL_ORDER
   long              position_ticket;  // DEAL_POSITION_ID
   datetime          time;             // DEAL_TIME
   char              type;             // DEAL_TYPE
   char              entry;            // DEAL_ENTRY
   string            symbol;           // DEAL_SYMBOL
   double            volume;           // DEAL_VOLUME
   double            price;            // DEAL_PRICE
```

© 2000-2025, MetaQuotes Ltd.

```
   double            profit;           // DEAL_PROFIT
   double            swap;             // DEAL_SWAP
   double            commission;       // DEAL_COMMISSION
   long              magic;            // DEAL_MAGIC
   char              reason;           // DEAL_REASON
  };
//--- Structure to store the trade: the order of members corresponds to the position i
struct Trade
  {
   datetime          time_in;          // entry time
   ulong             ticket;           // position ID
   char              type;             // buy or sell
   double            volume;           // volume
   string            symbol;           // symbol
   double            price_in;         // entry price
   datetime          time_out;         // exit time
   double            price_out;        // exit price
   double            commission;       // entry and exit commission
   double            swap;             // swap
   double            profit;           // profit or loss
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
   string filename=IntegerToString(AccountInfoInteger(ACCOUNT_LOGIN))+"_trades.sqlite"
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- create the DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   int deals_total=HistoryDealsTotal();
   PrintFormat("Deals in the trading history: %d ", deals_total);
//--- add deals to the table
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!InsertDeals(db))
      return;
//--- show the first 10 deals
   Deal deals[], deal;
   ArrayResize(deals, 10);
   int request=DatabasePrepare(db, "SELECT * FROM DEALS");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   int i;
   for(i=0; DatabaseReadBind(request, deal); i++)
     {
      if(i>=10)
         break;
      deals[i].ticket=deal.ticket;
      deals[i].order_ticket=deal.order_ticket;
      deals[i].position_ticket=deal.position_ticket;
      deals[i].time=deal.time;
      deals[i].type=deal.type;
      deals[i].entry=deal.entry;
      deals[i].symbol=deal.symbol;
      deals[i].volume=deal.volume;
      deals[i].price=deal.price;
      deals[i].profit=deal.profit;
      deals[i].swap=deal.swap;
      deals[i].commission=deal.commission;
      deals[i].magic=deal.magic;
      deals[i].reason=deal.reason;
     }
//--- print the deals
   if(i>0)
     {
      ArrayResize(deals, i);
      PrintFormat("The first %d deals:", i);
      ArrayPrint(deals);
     }

//--- delete request after use
   DatabaseFinalize(request);

//--- make sure that hedging system for open position management is used on the accou
   if((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)!=ACCOUNT_MARGI
     {
      //--- deals cannot be transformed to trades using a simple method through transa
      DatabaseClose(db);
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
      }

//--- now create the TRADES table based on the DEALS table
   if(!CreateTableTrades(db))
     {
      DatabaseClose(db);
      return;
     }
//--- fill in the TRADES table using an SQL query based on DEALS table data
   ulong start=GetMicrosecondCount();
   if(DatabaseTableExists(db, "DEALS"))
      //--- populate the TRADES table
      if(!DatabaseExecute(db, "INSERT INTO TRADES(TIME_IN,TICKET,TYPE,VOLUME,SYMBOL,PR
                              "SELECT "
                              "   d1.time as time_in,"
                              "   d1.position_id as ticket,"
                              "   d1.type as type,"
                              "   d1.volume as volume,"
                              "   d1.symbol as symbol,"
                              "   d1.price as price_in,"
                              "   d2.time as time_out,"
                              "   d2.price as price_out,"
                              "   d1.commission+d2.commission as commission,"
                              "   d2.swap as swap,"
                              "   d2.profit as profit "
                              "FROM DEALS d1 "
                              "INNER JOIN DEALS d2 ON d1.position_id=d2.position_id "
                              "WHERE d1.entry=0 AND d2.entry=1     "))
        {
         Print("DB: fillng the TRADES table failed with code ", GetLastError());
         return;
        }
   ulong transaction_time=GetMicrosecondCount()-start;

//--- show the first 10 deals
   Trade trades[], trade;
   ArrayResize(trades, 10);
   request=DatabasePrepare(db, "SELECT * FROM TRADES");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   for(i=0; DatabaseReadBind(request, trade); i++)
     {
      if(i>=10)
         break;
      trades[i].time_in=trade.time_in;
```

© 2000-2025, MetaQuotes Ltd.

```
      trades[i].ticket=trade.ticket;
      trades[i].type=trade.type;
      trades[i].volume=trade.volume;
      trades[i].symbol=trade.symbol;
      trades[i].price_in=trade.price_in;
      trades[i].time_out=trade.time_out;
      trades[i].price_out=trade.price_out;
      trades[i].commission=trade.commission;
      trades[i].swap=trade.swap;
      trades[i].profit=trade.profit;
     }
//--- print trades
   if(i>0)
     {
      ArrayResize(trades, i);
      PrintFormat("\r\nThe first %d trades:", i);
      ArrayPrint(trades);
      PrintFormat("Filling the TRADES table took %.2f milliseconds",double(transactio
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- close the database
   DatabaseClose(db);
  }
/*
Results:
   Deals in the trading history: 2741
   The first 10 deals:
       [ticket] [order_ticket] [position_ticket]              [time] [type] [entry] [s
   [0] 34429573              0                 0 2019.09.05 22:39:59      2       0 "
   [1] 34432127       51447238          51447238 2019.09.06 06:00:03      0       0 "U
   [2] 34432128       51447239          51447239 2019.09.06 06:00:03      1       0 "U
   [3] 34432450       51447565          51447565 2019.09.06 07:00:00      0       0 "E
   [4] 34432456       51447571          51447571 2019.09.06 07:00:00      1       0 "A
   [5] 34432879       51448053          51448053 2019.09.06 08:00:00      1       0 "U
   [6] 34432888       51448064          51448064 2019.09.06 08:00:00      0       0 "U
   [7] 34435147       51450470          51450470 2019.09.06 10:30:00      1       0 "E
   [8] 34435152       51450476          51450476 2019.09.06 10:30:00      0       0 "G
   [9] 34435154       51450479          51450479 2019.09.06 10:30:00      1       0 "E

   The first 10 trades:
                  [time_in] [ticket] [type] [volume] [symbol] [price_in]           [time
   [0] 2019.09.06 06:00:03 51447238      0  0.10000 "USDCAD"    1.32320 2019.09.06 18:
   [1] 2019.09.06 06:00:03 51447239      1  0.10000 "USDCHF"    0.98697 2019.09.06 18:
   [2] 2019.09.06 07:00:00 51447565      0  0.10000 "EURUSD"    1.10348 2019.09.09 03:
   [3] 2019.09.06 07:00:00 51447571      1  0.10000 "AUDUSD"    0.68203 2019.09.09 03:
   [4] 2019.09.06 08:00:00 51448053      1  0.10000 "USDCHF"    0.98701 2019.09.06 18:
   [5] 2019.09.06 08:00:00 51448064      0  0.10000 "USDJPY"  106.96200 2019.09.06 18:
```

© 2000-2025, MetaQuotes Ltd.

```
   [6] 2019.09.06 10:30:00 51450470      1  0.10000 "EURUSD"    1.10399 2019.09.06 14:
   [7] 2019.09.06 10:30:00 51450476      0  0.10000 "GBPUSD"    1.23038 2019.09.06 14:
   [8] 2019.09.06 10:30:00 51450479      1  0.10000 "EURJPY"  118.12000 2019.09.06 14:
   [9] 2019.09.06 10:30:00 51450480      0  0.10000 "GBPJPY"  131.65300 2019.09.06 14:
   Filling the TRADES table took 12.51 milliseconds
*/
//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(database, "DEALS"))
     {
      return(false);
     }
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID          INT KEY NOT NULL,"
                          "ORDER_ID    INT     NOT NULL,"
                          "POSITION_ID INT     NOT NULL,"
                          "TIME        INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "ENTRY       INT     NOT NULL,"
                          "SYMBOL      CHAR(10),"
                          "VOLUME      REAL,"
                          "PRICE       REAL,"
                          "PROFIT      REAL,"
                          "SWAP        REAL,"
                          "COMMISSION  REAL,"
                          "MAGIC       INT,"
                          "REASON      INT );"))
        {
         Print("DB: create the DEALS table failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database        |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
     {
      Print("Failed to drop the DEALS table  with code ", GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
     }
//--- the table has been successfully deleted
   return(true);
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database)
  {
//--- Auxiliary variables
   ulong    deal_ticket;         // deal ticket
   long     order_ticket;        // the ticket of the order by which the deal was exec
   long     position_ticket;     // ID of the position to which the deal belongs
   datetime time;                // deal execution time
   long     type ;               // deal type
   long     entry ;              // deal direction
   string   symbol;              // the symbol fro which the deal was executed
   double   volume;              // operation volume
   double   price;               // price
   double   profit;              // financial result
   double   swap;                // swap
   double   commission;          // commission
   long     magic;               // Magic number (Expert Advisor ID)
   long     reason;              // deal execution reason or source
//--- go through all deals and add them to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
// --- lock the database before executing transactions
   DatabaseTransactionBegin(database);
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      //--- add each deal to the table using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TIN
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G,
```

© 2000-2025, MetaQuotes Ltd.

```
                                        deal_ticket, order_ticket, position_ticket, tir
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%d with code %d", __FUNCTION__, deal_
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
         break;
        }
     }
//--- check for transaction execution errors
   if(failed)
     {
      //--- roll back all transactions and unlock the database
      DatabaseTransactionRollback(database);
      PrintFormat("%s: DatabaseExecute() failed with code %d", __FUNCTION__, GetLastEr
      return(false);
     }
//--- all transactions have been performed successfully - record changes and unlock th
   DatabaseTransactionCommit(database);
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the TRADES table                                         |
//+------------------------------------------------------------------+
bool CreateTableTrades(int database)
  {
//--- if the TRADES table already exists, delete it
   if(!DeleteTable(database, "TRADES"))
      return(false);
//--- check if the table exists
   if(!DatabaseTableExists(database, "TRADES"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE TRADES("
                          "TIME_IN     INT     NOT NULL,"
                          "TICKET      INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "VOLUME      REAL,"
                          "SYMBOL      CHAR(10),"
                          "PRICE_IN    REAL,"
                          "TIME_OUT    INT     NOT NULL,"
                          "PRICE_OUT   REAL,"
                          "COMMISSION  REAL,"
                          "SWAP        REAL,"
                          "PROFIT      REAL);"))
        {
         Print("DB: create the TRADES table failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
```

© 2000-2025, MetaQuotes Ltd.

```
    return(true);
   }
//+------------------------------------------------------------------+
```

另 见

DatabaseExecute、DatabaseFinalize

© 2000-2025, MetaQuotes Ltd.

# DatabaseReset

重置请求，比如调用DatabasePrepare()之后。

```
int  DatabaseReset(
   int  request      //在DatabasePrepare中接收的请求句柄
   );
```

参数

*request*

[in] DatabasePrepare()中获得的请求句柄。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) - 无效数据库句柄；
- 以ERR_DATABASE_ERROR (5601)开始的SQLite错误代码。

注意

DatabaseReset()函数用于通过不同的参数值多次执行请求。例如，当使用INSERT命令将数据批量添加到表格中时，应该为每个条目形成一组自定义的字段值。

与DatabasePrepare()不同，DatabaseReset() 调用不会通过SQL命令将字符串编译到新请求中，因此DatabaseReset()的执行速度要比DatabasePrepare()快得多。

如果在执行DatabaseRead()之后应该更改请求参数值，则DatabaseReset()与DatabaseBind()函数和/或DatabaseBindArray()一起使用。这意味着在设置新请求参数值之前（阻止DatabaseBind/DatabaseBindArray调用之前），应该调用DatabaseReset()进行重置。参数化请求本身应该使用DatabasePrepare()来创建。

与atabasePrepare()一样，DatabaseReset()不会发出数据库请求。当调用DatabaseRead()或DatabaseReadBind()时，进行直接请求执行。

如果通过调用DatabaseBind()/DatabaseBindArray()来设置参数值，则DatabaseReset()调用不会导致重置请求中的参数值，即参数保留其值。因此，只能更改单个参数值。调用DatabaseReset()后不需要重新设置所有请求参数。

使用DatabaseFinalize()移除的请求句柄不能传递给DatabaseReset()。这将导致错误。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建或打开数据库
   string filename="symbols.sqlite";
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE);
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");
//--- 如果SYMBOLS表格已存在，请将其删除
   if(DatabaseTableExists(db, "SYMBOLS"))
     {
      //--- 删除表格
      if(!DatabaseExecute(db, "DROP TABLE SYMBOLS"))
        {
         Print("Failed to drop table SYMBOLS with code ", GetLastError());
         DatabaseClose(db);
         return;
        }
     }
//--- 创建SYMBOLS表格
   if(!DatabaseExecute(db, "CREATE TABLE SYMBOLS("
                       "NAME            TEXT    NOT NULL,"
                       "DESCRIPTION     TEXT             ,"
                       "PATH            TEXT             ,"
                       "SPREAD          INT              ,"
                       "POINT           REAL    NOT NULL,"
                       "DIGITS          INT     NOT NULL,"
                       "JSON            BLOB );"))
     {
      Print("DB: ", filename, " create table failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- 显示SYMBOLS表格中所有字段列表
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(SYMBOLS)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(SYMBOLS)\") failed, error code=%d
      DatabaseClose(db);
      return;
     }

//--- 创建参数化请求，将交易品种添加到SYMBOLS表格中
   string sql="INSERT INTO SYMBOLS (NAME,DESCRIPTION,PATH,SPREAD,POINT,DIGITS,JSON)"
              " VALUES (?1,?2,?3,?4,?5,?6,?7);"; // 请求参数
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 检查所有交易品种并将其添加到SYMBOLS表格中
   int symbols=SymbolsTotal(false);
   bool request_error=false;
   DatabaseTransactionBegin(db);
   for(int i=0; i<symbols; i++)
     {
      //--- 在添加交易品种之前设置参数值
      ResetLastError();
      string symbol=SymbolName(i, false);
      if(!DatabaseBind(request, 0, symbol))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      //--- 如果之前的DatabaseBind()调用成功,则设置下一个参数
      if(!DatabaseBind(request, 1, SymbolInfoString(symbol, SYMBOL_DESCRIPTION)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 2, SymbolInfoString(symbol, SYMBOL_PATH)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 3, SymbolInfoInteger(symbol, SYMBOL_SPREAD)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 4, SymbolInfoDouble(symbol, SYMBOL_POINT)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 5, SymbolInfoInteger(symbol, SYMBOL_DIGITS)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 6, GetSymBolAsJson(symbol)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
```

© 2000-2025, MetaQuotes Ltd.

```
         request_error=true;
         break;
        }

      //--- 执行插入条目的请求并检查错误
      if(!DatabaseRead(request)&&(GetLastError()!=ERR_DATABASE_NO_MORE_DATA))
        {
         PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
      else
         PrintFormat("%d: added %s", i+1, symbol);
      //--- 在下一次参数更新之前重置请求
      if(!DatabaseReset(request))
        {
         PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
     } //--- 完成所有交易品种的检查

//--- 交易事务状态
   if(request_error)
     {
      PrintFormat("Table SYMBOLS: failed to add %d symbols", symbols);
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table SYMBOLS: added %d symbols",symbols);
     }

//--- 将SYMBOLS表格保存为CSV文件。
   string csv_filename="symbols.csv";
   if(DatabaseExport(db, "SELECT * FROM SYMBOLS", csv_filename,
                     DATABASE_EXPORT_HEADER|DATABASE_EXPORT_INDEX|DATABASE_EXPORT_QUOT
      Print("Database: table SYMBOLS saved in ", csv_filename);
   else
      Print("Database: DatabaseExport(\"SELECT * FROM SYMBOLS\") failed with code", Ge

//--- 关闭数据库文件并告知
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| 交易品种规格返回为JSON                                           |
//+------------------------------------------------------------------+
string GetSymBolAsJson(string symbol)
  {
//--- 缩进
   string indent1=Indent(1);
   string indent2=Indent(2);
   string indent3=Indent(3);
//---
   int digits=(int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   string json="{"+
               "\n"+indent1+"\"ConfigSymbols\":["+
               "\n"+indent2+"{"+
               "\n"+indent3+"\"Symbol\":\""+symbol+"\","+
               "\n"+indent3+"\"Path\":\""+SymbolInfoString(symbol, SYMBOL_PATH)+"\","+
               "\n"+indent3+"\"CurrencyBase\":\""+SymbolInfoString(symbol, SYMBOL_CURR
               "\n"+indent3+"\"CurrencyProfit\":\""+SymbolInfoString(symbol, SYMBOL_CU
               "\n"+indent3+"\"CurrencyMargin\":\""+SymbolInfoString(symbol, SYMBOL_CU
               "\n"+indent3+"\"ColorBackground\":\""+ColorToString((color)SymbolInfoIn
               "\n"+indent3+"\"Digits\":\""+IntegerToString(SymbolInfoInteger(symbol,
               "\n"+indent3+"\"Point\":\""+DoubleToString(SymbolInfoDouble(symbol, SYM
               "\n"+indent3+"\"TickBookDepth\":\""+IntegerToString(SymbolInfoInteger(s
               "\n"+indent3+"\"ChartMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"TradeMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"TradeCalcMode\":\""+IntegerToString(SymbolInfoInteger(s
               "\n"+indent3+"\"OrderMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"CalculationMode\":\""+IntegerToString(SymbolInfoIntege
               "\n"+indent3+"\"ExecutionMode\":\""+IntegerToString(SymbolInfoInteger(s
               "\n"+indent3+"\"ExpirationMode\":\""+IntegerToString(SymbolInfoInteger
               "\n"+indent3+"\"FillFlags\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"ExpirFlags\":\""+IntegerToString(SymbolInfoInteger(symb
               "\n"+indent3+"\"Spread\":\""+IntegerToString(SymbolInfoInteger(symbol,
               "\n"+indent3+"\"TickValue\":\""+StringFormat("%G", (SymbolInfoDouble(sy
               "\n"+indent3+"\"TickSize\":\""+StringFormat("%G", (SymbolInfoDouble(sym
               "\n"+indent3+"\"ContractSize\":\""+StringFormat("%G",(SymbolInfoDouble
               "\n"+indent3+"\"StopsLevel\":\""+IntegerToString(SymbolInfoInteger(symb
               "\n"+indent3+"\"VolumeMin\":\""+StringFormat("%G",(SymbolInfoDouble(sym
               "\n"+indent3+"\"VolumeMax\":\""+StringFormat("%G",(SymbolInfoDouble(sym
               "\n"+indent3+"\"VolumeStep\":\""+StringFormat("%G",(SymbolInfoDouble(sy
               "\n"+indent3+"\"VolumeLimit\":\""+StringFormat("%G",(SymbolInfoDouble(s
               "\n"+indent3+"\"SwapMode\":\""+IntegerToString(SymbolInfoInteger(symbol
               "\n"+indent3+"\"SwapLong\":\""+StringFormat("%G",(SymbolInfoDouble(symb
               "\n"+indent3+"\"SwapShort\":\""+StringFormat("%G",(SymbolInfoDouble(sym
               "\n"+indent3+"\"Swap3Day\":\""+IntegerToString(SymbolInfoInteger(symbol
               "\n"+indent2+"}"+
               "\n"+indent1+"]"+
               "\n}";
```

© 2000-2025, MetaQuotes Ltd.

```
   return(json);
  }
//+------------------------------------------------------------------+
//| 形成空格缩进                                                     |
//+------------------------------------------------------------------+
string Indent(const int number, const int characters=3)
  {
   int length=number*characters;
   string indent=NULL;
   StringInit(indent, length, ' ');
   return indent;
  }
/*
 结果:
  Database: symbols.sqlite opened successfully
  #| cid name        type notnull dflt_value pk
  -+-----------------------------------------
  1|   0 NAME        TEXT       1             0
  2|   1 DESCRIPTION TEXT       0             0
  3|   2 PATH        TEXT       0             0
  4|   3 SPREAD      INT        0             0
  5|   4 POINT       REAL       1             0
  6|   5 DIGITS      INT        1             0
  7|   6 JSON        BLOB       0             0
  1: added EURUSD
  2: added GBPUSD
  3: added USDCHF
  ...
  82: added USDCOP
  83: added USDARS
  84: added USDCLP
  Table SYMBOLS: added 84 symbols
  Database: table SYMBOLS saved in symbols.csv
  Database: symbols.sqlite created and closed
*/
```

另见

DatabasePrepare、DatabaseBind、DatabaseBindArray、DatabaseFinalize

© 2000-2025, MetaQuotes Ltd.

# DatabaseBind

在请求中设置一个参数值。

```
bool  DatabaseBind(
   int   request,       // DatabasePrepare中创建的请求句柄
   int   index,         // 请求中的参数索引
   T     value          // 简单类型参数的值
   );
```

参数

*request*

[in] DatabasePrepare()中创建的请求句柄。

*index*

[in] 应该为请求中的参数索引设置一个值。编号从0开始。

*value*

[in] 要设置的值。扩展类型：bool、char、uchar、short、ushart、int、uint、color、datetime、long、ulong、float、double、string。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) -不支持类型；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_NOT_READY (5128) -目前无法使用该函数发出请求。请求正在执行或已经完成。应该调用DatabaseReset()。

注意

该函数在SQL请求包含"?"或"?N"参数化值的情况下被使用，这里N表示参数索引（从1开始）。同时，DatabaseBind()中的参数索引从0开始。

例如：

```
INSERT INTO table VALUES (?,?,?)
SELECT * FROM table WHERE id=?
```

该函数可以在DatabasePrepare()中创建参数化请求或使用DatabaseReset()重置请求之后立即调用。

将该函数与DatabaseReset()一起使用，可以根据需要使用不同的参数值来多次执行请求。

该函数旨在使用简单的类型参数。如果应对数组检查参数，请使用DatabaseBindArray()函数。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   MqlTick ticks[];
//--- 记住收到报价的开始时间
   uint start=GetTickCount();
//--- 请求每日报价历史
   ulong to=TimeCurrent()*1000;
   ulong from=to-PeriodSeconds(PERIOD_D1)*1000;
   if(CopyTicksRange(_Symbol, ticks, COPY_TICKS_ALL, from, to)==-1)
     {
      PrintFormat("%s: CopyTicksRange(%s - %s) failed, error=%d",
                  _Symbol, TimeToString(datetime(from/1000)), TimeToString(datetime(to
      return;
     }
   else
     {
      //--- 接收多少报价，以及接收报价所花费的时间
      PrintFormat("%s: CopyTicksRange received %d ticks in %d ms (from %s to %s)",
                  _Symbol, ArraySize(ticks), GetTickCount()-start,
                  TimeToString(datetime(from/1000)), TimeToString(datetime(to/1000)));
     }

//--- 设置存储数据库的文件名称
   string filename=_Symbol+" "+TimeToString(datetime(from/1000))+" - "+TimeToString(da
   StringReplace(filename, ":", "."); // ":" character is not allowed in file names
//--- 在常规程序端文件夹中打开/创建数据库
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("Database: ", filename, " open failed with code ", GetLastError());
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");

//--- 创建TICKS表格
   if(!DatabaseExecute(db, "CREATE TABLE TICKS("
                       "SYMBOL             CHAR(10),"
                       "TIME               INT NOT NULL,"
                       "BID                REAL,"
                       "ASK                REAL,"
                       "LAST               REAL,"
                       "VOLUME             INT,"
                       "TIME_MSC           INT,"
                       "VOLUME_REAL        REAL);"))
     {
      Print("DB: ", filename, " create table TICKS failed with code ", GetLastError())
      DatabaseClose(db);
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 显示TICKS表格中所有字段列表
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(TICKS)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(TICKS)\") failed, error code=%d a
      DatabaseClose(db);
      return;
     }
//--- 创建参数化请求,将报价添加到TICKS表格中
   string sql="INSERT INTO TICKS (SYMBOL,TIME,BID,ASK,LAST,VOLUME,TIME_MSC,VOLUME_REAL
              " VALUES (?1,?2,?3,?4,?5,?6,?7,?8)"; // request parameters
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }
//--- 设置第一个请求参数的值
   DatabaseBind(request, 0, _Symbol);
//--- 记住将报价添加到TICKS表格的开始时间
   start=GetTickCount();
   DatabaseTransactionBegin(db);
   int total=ArraySize(ticks);
   bool request_error=false;
   for(int i=0; i<total; i++)
     {
      //--- 在添加条目之前设置其余参数的值
      ResetLastError();
      if(!DatabaseBind(request, 1, ticks[i].time))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      //--- 如果之前的DatabaseBind()调用成功,则设置下一个参数
      if(!request_error && !DatabaseBind(request, 2, ticks[i].bid))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 3, ticks[i].ask))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
```

© 2000-2025, MetaQuotes Ltd.

```
      break;
     }
   if(!request_error && !DatabaseBind(request, 4, ticks[i].last))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Tick #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
   if(!request_error && !DatabaseBind(request, 5, ticks[i].volume))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Tick #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
   if(!request_error && !DatabaseBind(request, 6, ticks[i].time_msc))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Tick #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
   if(!request_error && !DatabaseBind(request, 7, ticks[i].volume_real))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Tick #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }

   //--- 执行插入条目的请求并检查错误
   if(!request_error && !DatabaseRead(request) && (GetLastError()!=ERR_DATABASE_NO_
     {
      PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
      DatabaseFinalize(request);
      request_error=true;
      break;
     }
   //--- 在下一次参数更新之前重置请求
   if(!request_error && !DatabaseReset(request))
     {
      PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
      DatabaseFinalize(request);
      request_error=true;
      break;
     }
  } //--- 完成所有报价的检查
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 交易事务状态
   if(request_error)
     {
      PrintFormat("Table TICKS: failed to add %d ticks ", ArraySize(ticks));
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table TICKS: added %d ticks in %d ms",
                  ArraySize(ticks), GetTickCount()-start);
     }

//--- 关闭数据库文件并告知
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
  }
/*
 结果:
  EURUSD: CopyTicksRange received 268061 ticks in 47 ms (from 2020.03.18 12:40 to 2020
  Database: EURUSD 2020.03.18 12.40 - 2020.03.19 12.40.sqlite opened successfully
  #| cid name        type     notnull dflt_value pk
  -+-----------------------------------------------
  1|   0 SYMBOL      CHAR(10)       0              0
  2|   1 TIME        INT            1              0
  3|   2 BID         REAL           0              0
  4|   3 ASK         REAL           0              0
  5|   4 LAST        REAL           0              0
  6|   5 VOLUME      INT            0              0
  7|   6 TIME_MSC    INT            0              0
  8|   7 VOLUME_REAL REAL           0              0
  Table TICKS: added 268061 ticks in 797 ms
  Database: EURUSD 2020.03.18 12.40 - 2020.03.19 12.40.sqlite created and closed
  OnCalculateCorrelation=0.87 2020.03.19 13:00:  EURUSD vs GBPUSD  PERIOD_M30
*/
```

另见

DatabasePrepare、DatabaseReset、DatabaseRead、DatabaseBindArray

© 2000-2025, MetaQuotes Ltd.

# DatabaseBindArray

将数组设置为参数值。

```
bool  DatabaseBind(
   int   request,       // DatabasePrepare中创建的请求句柄
   int   index,         // 请求中的参数索引
   T&    array[]        // 数组设为参数值
   );
```

参数

*request*

[in] DatabasePrepare()中创建的请求句柄。

*index*

[in] 应该为请求中的参数索引设置一个值。编号从0开始。

*array[]*

[in] 要设置为请求参数值的数组。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) -不支持类型；
- ERR_ARRAY_BAD_SIZE (4011) -数组大小（以字节为单位）超过INT_MAX；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_NOT_READY (5128) -目前无法使用该函数发出请求（请求正被执行或已经完成，应该调用DatabaseReset）。

注意

该函数在SQL请求包含"?"或"?N"参数化值的情况下被使用，这里N表示参数索引（从1开始）。同时，DatabaseBindArray()中的参数索引从0开始。

例如：

```
INSERT INTO table VALUES (?,?,?)
```

该函数可以在DatabasePrepare()中创建参数化请求或使用DatabaseReset()重置请求之后立即调用。

将该函数与DatabaseReset()一起使用，可以根据需要使用不同的参数值来多次执行请求。

例如：

```
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 打开用于选择带有DAT扩展名的文件的对话框
   string selected_files[];
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!FileSelectDialog("Select files to download", NULL,
                        "Data files (*.dat)|*.dat|All files (*.*)|*.*",
                        FSD_ALLOW_MULTISELECT, selected_files, "tester.dat")>0)
     {
      Print("Files not selected. Exit");
      return;
     }
//--- 获取文件大小
   ulong filesize[];
   int filehandle[];
   int files=ArraySize(selected_files);
   ArrayResize(filesize, files);
   ZeroMemory(filesize);
   ArrayResize(filehandle, files);
   double total_size=0;
   for(int i=0; i<files; i++)
     {
      filehandle[i]=FileOpen(selected_files[i], FILE_READ|FILE_BIN);
      if(filehandle[i]!=INVALID_HANDLE)
        {
         filesize[i]=FileSize(filehandle[i]);
         //PrintFormat("%d, %s handle=%d %d bytes", i, selected_files[i], filehandle[i
         total_size+=(double)filesize[i];
        }
     }
//--- 检查常规文件大小
   if(total_size==0)
     {
      PrintFormat("Total files size is 0. Exit");
      return;
     }

//--- 在常规程序端文件夹中创建或打开数据库
   string filename="dat_files.sqlite";
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE);
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");
//--- 如果FILES表格已存在，请将其删除
   if(DatabaseTableExists(db, "FILES"))
     {
      //--- 删除表格
      if(!DatabaseExecute(db, "DROP TABLE FILES"))
        {
         Print("Failed to drop table FILES with code ", GetLastError());
```

© 2000-2025, MetaQuotes Ltd.

```
         DatabaseClose(db);
         return;
        }
     }
//--- 创建FILES表格
   if(!DatabaseExecute(db, "CREATE TABLE FILES("
                       "NAME           TEXT NOT NULL,"
                       "SIZE           INT  NOT NULL,"
                       "PERCENT_SIZE   REAL NOT NULL,"
                       "DATA           BLOB NOT NULL);"))
     {
      Print("DB: failed to create table FILES with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- 显示FILES表格中所有字段列表
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(FILES)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(FILES)\") failed, error code=%d a
      DatabaseClose(db);
      return;
     }

//--- 创建参数化请求，将文件添加到FILES表格中
   string sql="INSERT INTO FILES (NAME,SIZE,PERCENT_SIZE,DATA)"
              " VALUES (?1,?2,?3,?4);"; // 请求参数
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }

//--- 检查所有文件并将其添加到FILES表格中
   bool request_error=false;
   DatabaseTransactionBegin(db);
   int count=0;
   uint size;
   for(int i=0; i<files; i++)
     {
      if(filehandle[i]!=INVALID_HANDLE)
        {
         char data[];
         size=FileReadArray(filehandle[i], data);
         if(size==0)
           {
            PrintFormat("FileReadArray(%s) failed with code %d", selected_files[i], Ge
```

© 2000-2025, MetaQuotes Ltd.

```
            continue;
           }

      count++;
      //--- 在将文件添加到表格之前设置参数值
      if(!DatabaseBind(request, 0, selected_files[i]))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 1, size))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 2, double(size)*100./total_size))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBindArray(request, 3, data))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      //--- 执行插入条目的请求并检查错误
      if(!DatabaseRead(request)&&(GetLastError()!=ERR_DATABASE_NO_MORE_DATA))
        {
         PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
      else
         PrintFormat("%d. %s: %d bytes", count, selected_files[i],size);
      //--- 在下一次参数更新之前重置请求
      if(!DatabaseReset(request))
        {
         PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 交易事务状态
   if(request_error)
     {
      PrintFormat("Table FILES: failed to add %d files", count);
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table FILES: added %d files", count);
     }

//--- 关闭数据库文件并告知
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
  }
```

另 见

DatabasePrepare、DatabaseReset、DatabaseRead、DatabaseBind

© 2000-2025, MetaQuotes Ltd.

# DatabaseRead

作为请求结果，移到下一个条目。

```
bool  DatabaseRead(
   int  request      //在DatabasePrepare中接收的请求句柄
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) - 没有指定表格名称（空字符串或NULL）；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -内部数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误；
- ERR_DATABASE_NO_MORE_DATA (5126) -不存在表格（没有错误，正常完成）。

另见

DatabasePrepare, DatabaseReadBind

© 2000-2025, MetaQuotes Ltd.

# DatabaseReadBind

移动到下一条记录，并从中将数据读入结构。

```
bool  DatabaseReadBind(
   int     request,           //DatabasePrepare中创建的请求句柄
   void&   struct_object      //对读取记录结构的引用
   );
```

参数

*request*

[in] DatabasePrepare()中创建的请求句柄。

*struct_object*

[out] 要读入当前记录中数据的结构的引用。该结构应该只有数值类型和/或字符串（不允许数组）作为成员，并且不能是衍生体。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INVALID_PARAMETER (4003) - 没有指定表格名称（空字符串或NULL）；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -内部数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误；
- ERR_DATABASE_NO_MORE_DATA (5126) -不存在表格（没有错误，正常完成）。

注意

*struct_object*结构中的字段数不应超过DatabaseColumnsCount()。如果*struct_object*结构中的字段数小于记录中的字段数，则执行部分读取。使用相应的DatabaseColumnText()、DatabaseColumnInteger()等函数可以明确获取剩余的数据。

示例：

```
struct Person
  {
   int               id;
   string            name;
   int               age;
   string            address;
   double            salary;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int db;
   string filename="company.sqlite";
```

© 2000-2025, MetaQuotes Ltd.

```
//--- open
   db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE |DATABASE_
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- if the table COMPANY exists then drop the table
   if(DatabaseTableExists(db, "COMPANY"))
     {
      //--- delete the table
      if(!DatabaseExecute(db, "DROP TABLE COMPANY"))
        {
         Print("Failed to drop table COMPANY with code ", GetLastError());
         DatabaseClose(db);
         return;
        }
     }
//--- create table
   if(!DatabaseExecute(db, "CREATE TABLE COMPANY("
                       "ID INT PRIMARY KEY     NOT NULL,"
                       "NAME           TEXT    NOT NULL,"
                       "AGE            INT     NOT NULL,"
                       "ADDRESS        CHAR(50),"
                       "SALARY         REAL );"))
     {
      Print("DB: ", filename, " create table failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }

//--- insert data
   if(!DatabaseExecute(db, "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (1
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'A
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, 'T
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, 'M
     {
      Print("DB: ", filename, " insert failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }

//--- prepare the request
   int request=DatabasePrepare(db, "SELECT * FROM COMPANY WHERE SALARY>15000");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
```

© 2000-2025, MetaQuotes Ltd.

```
     }
//--- print records
   Person person;
   Print("Persons with salary > 15000:");
   for(int i=0; DatabaseReadBind(request, person); i++)
      Print(i, ":  ", person.id, " ", person.name, " ", person.age, " ", person.addres
//--- delete request after use
   DatabaseFinalize(request);

   Print("Some statistics:");
//--- prepare new request about total salary
   request=DatabasePrepare(db, "SELECT SUM(SALARY) FROM COMPANY");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   while(DatabaseRead(request))
     {
      double total_salary;
      DatabaseColumnDouble(request, 0, total_salary);
      Print("Total salary=", total_salary);
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- prepare new request about average salary
   request=DatabasePrepare(db, "SELECT AVG(SALARY) FROM COMPANY");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      ResetLastError();
      DatabaseClose(db);
      return;
     }
   while(DatabaseRead(request))
     {
      double aver_salary;
      DatabaseColumnDouble(request, 0, aver_salary);
      Print("Average salary=", aver_salary);
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- close database
   DatabaseClose(db);
  }
//+------------------------------------------------------------------
```

© 2000-2025, MetaQuotes Ltd.

```
/*
Output:
Persons with salary > 15000:
0:  1 Paul 32 California 25000.0
1:  3 Teddy 23 Norway 20000.0
2:  4 Mark 25 Rich-Mond  65000.0
Some statistics:
Total salary=125000.0
Average salary=31250.0
*/
```

另见

DatabasePrepare, DatabaseRead

© 2000-2025, MetaQuotes Ltd.

# DatabaseFinalize

移除在DatabasePrepare()中创建的请求参数。

```
void  DatabaseFinalize(
   int  request      //在DatabasePrepare中接收的请求句柄
   );
```

参数

*database*

[in] 在DatabasePrepare()中接收的请求句柄。

返回值

无。

注意

如果句柄无效，函数设置ERR_DATABASE_INVALID_HANDLE错误。您可以使用GetLastError()检查错误。

另见

DatabasePrepare、DatabaseExecute

© 2000-2025, MetaQuotes Ltd.

# DatabaseTransactionBegin

开始事务执行。

```
bool  DatabaseTransactionBegin(
   int  database      // 在DatabaseOpen中接收的数据库句柄
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) - 重要运行时错误；
- ERR_INVALID_PARAMETER (4003) - sql参数包含空字符串；
- ERR_NOT_ENOUGH_MEMORY (4004) - 内存不足；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误。

注意

DatabaseTransactionBegin()函数应该在事务执行之前调用。任何事务都应该以调用DatabaseTransactionBegin()开始，以调用DatabaseTransactionCommit()结束。

**Example:**

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
   string filename=AccountInfoString(ACCOUNT_SERVER) +"_"+IntegerToString(AccountInfo
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(db, "DEALS"))
     {
      DatabaseClose(db);
      return;
     }
//--- create the DEALS table
   if(!CreateTableDeals(db))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       DatabaseClose(db);
       return;
      }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   int deals_total=HistoryDealsTotal();
   PrintFormat("Deals in the trading history: %d ", deals_total);

//--- measure the transaction execution speed using DatabaseTransactionBegin/DatabaseT
   ulong start=GetMicrosecondCount();
   bool fast_transactions=true;
   InsertDeals(db, fast_transactions);
   double fast_transactions_time=double(GetMicrosecondCount()-start)/1000;
   PrintFormat("Transations WITH    DatabaseTransactionBegin/DatabaseTransactionCommit

//--- delete the DEALS table, and then create it again
   if(!DeleteTable(db, "DEALS"))
      {
       DatabaseClose(db);
       return;
      }
//--- create a new DEALS table
   if(!CreateTableDeals(db))
      {
       DatabaseClose(db);
       return;
      }

//--- test again, this time without using DatabaseTransactionBegin/DatabaseTransactio
   fast_transactions=false;
   start=GetMicrosecondCount();
   InsertDeals(db, fast_transactions);
   double slow_transactions_time=double(GetMicrosecondCount()-start)/1000;
   PrintFormat("Transations WITHOUT DatabaseTransactionBegin/DatabaseTransactionCommit
//--- report gain in time
   PrintFormat("Use of DatabaseTransactionBegin/DatabaseTransactionCommit provided acc
//--- close the database
   DatabaseClose(db);
  }
/*
Results:
   Deals in the trading history: 2737
   Transations WITH    DatabaseTransactionBegin/DatabaseTransactionCommit: time=48.5 m
   Transations WITHOUT DatabaseTransactionBegin/DatabaseTransactionCommit: time=25818.
   Use of DatabaseTransactionBegin/DatabaseTransactionCommit provided acceleration by
```

© 2000-2025, MetaQuotes Ltd.

```
*/
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database        |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
     {
      Print("Failed to drop table with code ", GetLastError());
      return(false);
     }
//--- the table has been successfully deleted
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID          INT KEY NOT NULL,"
                          "ORDER_ID    INT     NOT NULL,"
                          "POSITION_ID INT     NOT NULL,"
                          "TIME        INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "ENTRY       INT     NOT NULL,"
                          "SYMBOL      CHAR(10),"
                          "VOLUME      REAL,"
                          "PRICE       REAL,"
                          "PROFIT      REAL,"
                          "SWAP        REAL,"
                          "COMMISSION  REAL,"
                          "MAGIC       INT,"
                          "REASON      INT );"))
        {
         Print("DB: create the table DEALS failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database, bool begintransaction=true)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- Auxiliary variables
   ulong    deal_ticket;          // deal ticket
   long     order_ticket;         // the ticket of the order by which the deal was exec
   long     position_ticket;      // ID of the position to which the deal belongs
   datetime time;                 // deal execution time
   long     type ;                // deal type
   long     entry ;               // deal direction
   string   symbol;               // the symbol fro which the deal was executed
   double   volume;               // operation volume
   double   price;                // price
   double   profit;               // financial result
   double   swap;                 // swap
   double   commission;           // commission
   long     magic;                // Magic number
   long     reason;               // deal execution reason or source
//--- go through all deals and add to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
//--- if fast transaction performance method is used
   if(begintransaction)
     {
      // --- lock the database before executing transactions
      DatabaseTransactionBegin(database);
     }
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      //--- add each deal using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TIM
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G,
                                       deal_ticket, order_ticket, position_ticket, tim
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%dwith code %d", __FUNCTION__, deal_t
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
```

© 2000-2025, MetaQuotes Ltd.

```
         break;
        }
      }
//--- check for transaction execution errors
   if(failed)
     {
      //--- if fast transaction performance method is used
      if(begintransaction)
        {
         //--- roll back all transactions and unlock the database
         DatabaseTransactionRollback(database);
        }
      Print("%s: DatabaseExecute() failed with code ", __FUNCTION__, GetLastError());
      return(false);
     }
//--- if fast transaction performance method is used
   if(begintransaction)
     {
      //--- all transactions have been performed successfully - record changes and unl
      DatabaseTransactionCommit(database);
     }
//--- successful completion
   return(true);
  }
//+------------------------------------------------------------------+
```

另见

DatabaseExecute、DatabasePrepare、DatabaseTransactionCommit、DatabaseTransactionRollback

© 2000-2025, MetaQuotes Ltd.

# DatabaseTransactionCommit

完成事务执行。

```
bool  DatabaseTransactionCommit(
   int  database      // 在DatabaseOpen中接收的数据库句柄
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) - 重要运行时错误；
- ERR_INVALID_PARAMETER (4003) - sql参数包含空字符串；
- ERR_NOT_ENOUGH_MEMORY (4004) - 内存不足；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -无效数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误。

注意

DatabaseTransactionCommit()函数完成调用DatabaseBeginTransaction()函数后执行的所有事务。任何事务都应该以调用DatabaseTransactionBegin()开始，以调用DatabaseTransactionCommit()结束。

另见

DatabaseExecute、DatabasePrepare、DatabaseTransactionBegin、DatabaseTransactionRollback

© 2000-2025, MetaQuotes Ltd.

# DatabaseTransactionRollback

回滚事务。

```
bool  DatabaseTransactionRollback(
   int  database      // 在DatabaseOpen中接收的数据库句柄
   );
```

参数

*database*

[in] 在DatabaseOpen()中接收的数据库句柄。

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_INTERNAL_ERROR (4001) - 重要运行时错误；
- ERR_INVALID_PARAMETER (4003) - sql参数包含空字符串；
- ERR_NOT_ENOUGH_MEMORY (4004) - 内存不足；
- ERR_WRONG_STRING_PARAMETER (5040) -将请求转换为UTF-8字符串时出现错误；
- ERR_DATABASE_INTERNAL (5120) -内部数据库错误；
- ERR_DATABASE_INVALID_HANDLE (5121) -内部数据库句柄；
- ERR_DATABASE_EXECUTE (5124) - 请求执行错误；

注意

DatabaseTransactionRollback()调用取消调用DatabaseTransactionBegin()函数后执行的所有事务。在执行事务发生错误时，需要DatabaseTransactionRollback()函数回滚数据库中的更改。

另见

DatabaseExecute、DatabasePrepare、DatabaseTransactionBegin、DatabaseTransactionCommit

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnsCount

获取请求中的字段数。

```
int  DatabaseColumnsCount(
   int  request      //在DatabasePrepare中接收的请求句柄
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

返回值

字段数量，错误情况下为-1。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) - 无效请求句柄。

注意

不需要调用DatabaseRead()函数来获取在DatabasePrepare()中创建的请求的字段数。对于其余的DatabaseColumnXXX()函数，应该初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseFinalize、DatabaseClose

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnName

按索引获取字段名。

```
bool  DatabaseColumnName(
   int       request,      // 在DatabasePrepare中接收的请求句柄
   int       column,       // 请求中的字段索引
   string&   name          //用于接收字段名的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*name*

[out] 用于编写字段名的变量。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnType

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnType

按索引获取字段类型。

```
ENUM_DATABASE_FIELD_TYPE  DatabaseColumnType(
   int  request,     // 在DatabasePrepare中接收的请求句柄
   int  column       //请求中的字段索引
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

返回值

从ENUM_DATABASE_FIELD_TYPE枚举返回字段类型。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) - 'column'索引超过DatabaseColumnsCount() -1.

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

**ENUM_DATABASE_FIELD_TYPE**

| ID | 描述 |
|---|---|
| DATABASE_FIELD_TYPE_INVALID | 获取类型错误，可使用int GetLastError()获取错误代码 |
| DATABASE_FIELD_TYPE_INTEGER | 整型 |
| DATABASE_FIELD_TYPE_FLOAT | 真实型 |
| DATABASE_FIELD_TYPE_TEXT | 字符串类型 |
| DATABASE_FIELD_TYPE_BLOB | 二进制类型 |
| DATABASE_FIELD_TYPE_NULL | 特殊NULL类型 |

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnSize

获取字段大小（以字节为单位）。

```
int  DatabaseColumnSize(
   int  request,     // 在DatabasePrepare中接收的请求句柄
   int  column       //请求中的字段索引
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

返回值

如果成功，返回字段大小（以字节为单位），否则为-1。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request（请求）'初步发出至少一个DatabaseRead()调用时，才能获取该值。

另见

DatabasePrepare、DatabaseColumnBlob、DatabaseColumnsCount、DatabaseColumnName、DatabaseColumnType

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnText

从当前记录中获取作为字符串的字段值。

```
bool  DatabaseColumnText(
   int       request,      // 在DatabasePrepare中接收的请求句柄
   int       column,       // 请求中的字段索引
   string&   value         //用于接收值的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*value*

[out] 用于编写字段值的变量的引用。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

若要从下一个记录读取该值，请初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnType、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnInteger

从当前记录中获取int类型的值。

```
bool  DatabaseColumnInteger(
   int    request,      // 在DatabasePrepare中接收的请求句柄
   int    column,       // 请求中的字段索引
   int&   value         // 用于接收值的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*value*

[out] 用于编写字段值的变量的引用。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

若要从下一个记录读取该值，请初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnType、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnLong

从当前记录中获取long类型的值。

```
bool  DatabaseColumnLong(
   int     request,      // 在DatabasePrepare中接收的请求句柄
   int     column,       // 请求中的字段索引
   long&   value         //用于接收值的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*value*

[out] 用于编写字段值的变量的引用。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

若要从下一个记录读取该值，请初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnType、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnDouble

从当前记录中获取double类型的值。

```
bool  DatabaseColumnDouble(
   int       request,      // 在DatabasePrepare中接收的请求句柄
   int       column,       // 请求中的字段索引
   double&   value         // 用于接收值的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*value*

[out] 用于编写字段值的变量的引用。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount()-1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

若要从下一个记录读取该值，请初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseColumnsCount、DatabaseColumnType、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# DatabaseColumnBlob

从当前记录中获取作为数组的字段值。

```
bool  DatabaseColumnBlob(
   int     request,      // 在DatabasePrepare中接收的请求句柄
   int     column,       // 请求中的字段索引
   void&   data[]        // 用于接收值的变量的引用
   );
```

参数

*request*

[in] 在DatabasePrepare()中接收的请求句柄。

*column*

[in] 请求中的字段索引。字段编号从零开始，不能超过 DatabaseColumnsCount() - 1。

*data[]*

[out] 用于编写字段值的数组的引用。

返回值

如果成功返回true，否则返回false。要获得错误代码，请使用GetLastError()，可能回应：

- ERR_DATABASE_INVALID_HANDLE (5121) -无效请求句柄；
- ERR_DATABASE_NO_MORE_DATA (5126) -'column'索引超过DatabaseColumnsCount() -1。

注意

只有在为'request (请求)'初步发出至少一个DatabaseRead()调用时，才能获取该值。

若要从下一个记录读取该值，请初步调用DatabaseRead()。

另见

DatabasePrepare、DatabaseColumnSize、DatabaseColumnsCount、DatabaseColumnType、DatabaseColumnName

© 2000-2025, MetaQuotes Ltd.

# 使用DirectX

DirectX 11功能和着色器旨在直接在价格图表上实现3D可视化。

创建3D图形需要一个具有所需图像大小的图形环境(DXContextCreate)。此外，还需要准备顶点和索引缓冲区(DXBufferCreate)，以及创建顶点和像素着色器(DXShaderCreate)。这足以显示彩色图形。

下一级图形需要输入(DXInputSet)以将其他渲染参数传递到着色器。这可以设置相机和3D对象的位置、描述光源并实现鼠标和键盘控制。

因此，内置的MQL5函数可以使您在MetaTrader 5中直接创建动画3D图表，无需第三方工具。显卡应支持DX 11和Shader Model5.0，以便使函数正常工作。

若要开始使用该函数库，只需阅读文章如何在MetaTrader 5中使用DirectX创建3D图形。

| 函数 | 操作 |
| --- | --- |
| DXContextCreate | 创建图形环境，以渲染指定大小的画帧 |
| DXContextSetSize | 更改在DXContextCreate()中创建的图形环境的画帧大小 |
| DXContextSetSize | 获取在DXContextCreate()中创建的图形环境的画帧大小 |
| DXContextClearColors | 为渲染缓冲区的所有像素设置指定的颜色 |
| DXContextClearDepth | 清除深度缓冲区 |
| DXContextGetColors | 从图形环境获取具有指定大小和偏移量的图像 |
| DXContextGetDepth | 获取渲染画帧的深度缓冲区 |
| DXBufferCreate | 根据数据数组创建指定类型的缓冲区 |
| DXTextureCreate | 从传递图像中剪切的指定大小的矩形中创建2D结构 |
| DXInputCreate | 创建着色器输入 |
| DXInputSet | 设置着色器输入 |
| DXShaderCreate | 创建指定类型的着色器 |
| DXShaderSetLayout | 设置顶点着色器的顶点布局 |
| DXShaderInputsSet | 设置着色器输入 |
| DXShaderTexturesSet | 设置着色器结构 |
| DXDraw | 渲染在DXBufferSet()中设置的顶点缓冲区的顶点 |
| DXDrawIndexed | 渲染由DXBufferSet()的索引缓冲区描述的图形原语 |
| DXPrimiveTopologySet | 设置使用DXDraw Indexed()渲染的原语类型 |
| DXBufferSet | 设置当前渲染的缓冲区 |
| DXShaderSet | 设置渲染的着色器 |
| DXHandleType | 返回句柄类型 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 操作 |
| --- | --- |
| DXRelease | 释放句柄 |

© 2000-2025, MetaQuotes Ltd.

# DXContextCreate

创建图形环境，以渲染指定大小的画帧。

```
int  DXContextCreate(
   uint  width,      // 像素宽度
   uint  height      // 像素高度
   );
```

参数

*width*

[in] 画帧宽度（以像素为单位）。

*height*

[in] 画帧高度（以像素为单位）。

返回值

在出现错误时，用于创建环境或者INVALID_HANDLE的句柄。若要接收错误代码，应该调用GetLastError()函数。

注意

使用DXBufferCreate、DXInputCreate、DXShaderCreate和DXTextureCreate函数创建的所有图形对象只可用在其创建的图形环境中。

随后可将画帧大小更改为DXContextSetSize()。

创建的不再使用的句柄应该由DXRelease()函数明确释放。

© 2000-2025, MetaQuotes Ltd.

# DXContextSetSize

更改在DXContextCreate()中创建的图形环境的画帧大小。

```
bool  DXContextSetSize(
   int    context,       // 图形环境句柄
   uint&  width,         // 像素宽度
   uint&  height         // 像素高度
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*width*

[in] 画帧宽度（以像素为单位）。

*height*

[in] 画帧高度（以像素为单位）。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

图像环境的画帧大小应该只能在画帧渲染之间改变。

© 2000-2025, MetaQuotes Ltd.

# DXContextGetSize

获取在DXContextCreate()中创建的图形环境的画帧大小。

```
bool  DXContextGetSize(
   int     context,       // 图形环境句柄
   uint&   width,         // 像素宽度
   uint&   height         // 像素高度
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*width*

[out] 画帧宽度（以像素为单位）。

*height*

[out] 画帧高度（以像素为单位）。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

© 2000-2025, MetaQuotes Ltd.

# DXContextClearColors

为渲染缓冲区的所有像素设置指定的颜色。

```
bool  DXContextClearColors(
   int              context,       // 图形环境句柄
   const DXVector&  color          // 颜色
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*color*

[in] 渲染颜色。

返回值

如果执行成功，返回true，否则 -false。若要接收错误代码，应该调用GetLastError()函数。

注意

渲染下一个画帧之前，可以使用DXContextClearColors()函数来清除颜色缓冲区。

© 2000-2025, MetaQuotes Ltd.

# DXContextClearDepth

清除深度缓冲区。

```
bool  DXContextClearDepth(
   int  context      // 图形环境句柄
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

渲染下一个画帧之前，可以使用DXContextClearDepth()函数来清除深度缓冲区。

© 2000-2025, MetaQuotes Ltd.

# DXContextGetColors

从图形环境获取具有指定大小和偏移量的图像。

```
bool  DXContextGetColors(
   int    context,                    // 图形环境句柄
   uint&  image[],                    // 图像像素数组
   int    image_width=WHOLE_ARRAY,    // 图像宽度（以像素为单位）
   int    image_height=WHOLE_ARRAY,   // 图像高度（以像素为单位）
   int    image_offset_x=0,           // X偏移量
   int    image_offset_y=0            // Y偏移量
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*image*

[out] *image_width***image_height*像素数组（以ARGB格式）。

*image_width=WHOLE_ARRAY*

[in] 图像宽度（以像素为单位）。

*image_height=WHOLE_ARRAY*

[in] 图像高度（以像素为单位）。

*image_offset_x=0*

[in] X偏移量。

*image_offset_y=0*

[in] Y偏移量。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

© 2000-2025, MetaQuotes Ltd.

# DXContextGetDepth

获取渲染画帧的深度缓冲区。

```
bool  DXContextGetDepth(
   int     context,       // 图形环境句柄
   float&  image[]        // 深度值数组
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*image*

[out] 渲染画帧深度缓冲区值的数组。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

返回的缓冲区包含能够以相对单位 (从0.0到1.0)在DXContextGetColors()中获得的渲染画帧的每个像素的深度。

© 2000-2025, MetaQuotes Ltd.

# DXBufferCreate

根据数据数组创建指定类型的缓冲区。

```
int  DXBufferCreate(
   int                  context,              // 图形环境句柄
   ENUM_DX_BUFFER_TYPE  buffer_type,          // 已创建缓冲区的类型
   const void&          data[],               // 缓存数据
   uint                 start=0,              // 初始索引
   uint                 count=WHOLE_ARRAY     // 元素数
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*buffer_type*

[in] 来自ENUM_DX_BUFFER_TYPE枚举的缓冲类型。

*data[]*

[in] 用于创建缓冲区的数据。

*start*

[in] 数组第一元素索引，数组值自此开始用来创建缓冲区。默认情况下，数据取自数据的开头。

*count*

[in] 值数。默认情况下，使用整个数组(count=WHOLE_ARRAY)。

返回值

在出现错误时，用于创建缓冲区或者INVALID_HANDLE的句柄。若要接收错误代码，应该调用GetLastError()函数。

注意

对于索引缓冲区，*data[]*数组应该是'uint'类型，而顶点缓冲区接收描述顶点的结构数组。

创建的不再使用的句柄应该由DXRelease()函数明确释放。

ENUM_DX_BUFFER_TYPE

| ID | 值 | 描述 |
|---|---|---|
| DX_BUFFER_VERTEX | 1 | 顶点缓冲区 |
| DX_BUFFER_INDEX | 2 | 索引缓冲区 |

© 2000-2025, MetaQuotes Ltd.

# DXTextureCreate

从传递图像中剪切的指定大小的矩形中创建2D结构。

```
int  DXTextureCreate(
   int              context,          // 图形环境句柄
   ENUM_DX_FORMAT   format,           // 像素颜色格式
   uint             width,            // 源图像宽度
   uint             height,           // 源图像高度
   const void&      data[],           // 源图像像素数组
   uint             data_x,           // 用于创建结构的矩形X坐标
   uint             data_y,           // 用于创建结构的矩形Y坐标
   uint             data_width,       // 用于创建结构的矩形宽度
   uint             data_height       // 用于创建结构的矩形高度
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*format*

[in] 从ENUM_DX_FORMAT枚举设置的像素颜色格式。

*width*

[in] 结构所基于的图像宽度。

*height*

[in] 结构所基于的图像高度。

*data*

[in] 结构所基于的图像像素数组。

*data_x*

[in] 用于创建结构的矩形X坐标（偏移量）。

*data_y*

[in] 用于创建结构的矩形Y坐标（偏移量）。

*data_width*

[in] 用于创建结构的矩形宽度。

*data_height*

[in] 用于创建结构的矩形高度。

返回值

在出现错误时的结构句柄或INVALID_HANDLE。若要接收错误代码，应该调用GetLastError()函数。

注意

创建的不再使用的句柄应该由DXRelease()函数明确释放。

© 2000-2025, MetaQuotes Ltd.

ENUM_DX_FORMAT

| ID | 值 | 匹配 DXGI_FORMAT |
|---|---|---|
| DX_FORMAT_UNKNOWN | 0 | DXGI_FORMAT_UNKNOWN |
| DX_FORMAT_R32G32B32A32_TYPELESS | 1 | DXGI_FORMAT_R32G32B32A32_TYPELESS |
| DX_FORMAT_R32G32B32A32_FLOAT | 2 | DXGI_FORMAT_R32G32B32A32_FLOAT |
| DX_FORMAT_R32G32B32A32_UINT | 3 | DXGI_FORMAT_R32G32B32A32_UINT |
| DX_FORMAT_R32G32B32A32_SINT | 4 | DXGI_FORMAT_R32G32B32A32_SINT |
| DX_FORMAT_R32G32B32_TYPELESS | 5 | DXGI_FORMAT_R32G32B32_TYPELESS |
| DX_FORMAT_R32G32B32_FLOAT | 6 | DXGI_FORMAT_R32G32B32_FLOAT |
| DX_FORMAT_R32G32B32_UINT | 7 | DXGI_FORMAT_R32G32B32_UINT |
| DX_FORMAT_R32G32B32_SINT | 8 | DXGI_FORMAT_R32G32B32_SINT |
| DX_FORMAT_R16G16B16A16_TYPELESS | 9 | DXGI_FORMAT_R16G16B16A16_TYPELESS |
| DX_FORMAT_R16G16B16A16_FLOAT | 10 | DXGI_FORMAT_R16G16B16A16_FLOAT |
| DX_FORMAT_R16G16B16A16_UNORM | 11/ | DXGI_FORMAT_R16G16B16A16_UNORM |
| DX_FORMAT_R16G16B16A16_UINT | 12/ | DXGI_FORMAT_R16G16B16A16_UINT |
| DX_FORMAT_R16G16B16A16_SNORM | 13 | DXGI_FORMAT_R16G16B16A16_SNORM |
| DX_FORMAT_R16G16B16A16_SINT | 14/ | DXGI_FORMAT_R16G16B16A16_SINT |
| DX_FORMAT_R32G32_TYPELESS | 15 | DXGI_FORMAT_R32G32_TYPELESS |
| DX_FORMAT_R32G32_FLOAT | 16/ | DXGI_FORMAT_R32G32_FLOAT |
| DX_FORMAT_R32G32_UINT | 17 | DXGI_FORMAT_R32G32_UINT |
| DX_FORMAT_R32G32_SINT | 18 | DXGI_FORMAT_R32G32_SINT |
| DX_FORMAT_R32G8X24_TYPELESS | 19/ | DXGI_FORMAT_R32G8X24_TYPELESS |
| DX_FORMAT_D32_FLOAT_S8X24_UINT | 20 | DXGI_FORMAT_D32_FLOAT_S8X24_UINT |
| DX_FORMAT_R32_FLOAT_X8X24_TYPELESS | 21 | DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS |
| DX_FORMAT_X32_TYPELESS_G8X24_UINT | 22 | DXGI_FORMAT_X32_TYPELESS_G8X24_UINT |
| DX_FORMAT_R10G10B10A2_TYPELESS | 23/ | DXGI_FORMAT_R10G10B10A2_TYPELESS |
| DX_FORMAT_R10G10B10A2_UNORM | 24/ | DXGI_FORMAT_R10G10B10A2_UNORM |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 匹配 DXGI_FORMAT |
|---|---|---|
| DX_FORMAT_R10G10B10A2_UINT | 25/ | DXGI_FORMAT_R10G10B10A2_UINT |
| DX_FORMAT_R11G11B10_FLOAT | 26/ | DXGI_FORMAT_R11G11B10_FLOAT |
| DX_FORMAT_R8G8B8A8_TYPELESS | 27/ | DXGI_FORMAT_R8G8B8A8_TYPELESS |
| DX_FORMAT_R8G8B8A8_UNORM | 28 | DXGI_FORMAT_R8G8B8A8_UNORM |
| DX_FORMAT_R8G8B8A8_UNORM_SRGB | 29 | DXGI_FORMAT_R8G8B8A8_UNORM_SRGB |
| DX_FORMAT_R8G8B8A8_UINT | 30 | DXGI_FORMAT_R8G8B8A8_UINT |
| DX_FORMAT_R8G8B8A8_SNORM | 31 | DXGI_FORMAT_R8G8B8A8_SNORM |
| DX_FORMAT_R8G8B8A8_SINT | 32 | DXGI_FORMAT_R8G8B8A8_SINT |
| DX_FORMAT_R16G16_TYPELESS | 33 | DXGI_FORMAT_R16G16_TYPELESS |
| DX_FORMAT_R16G16_FLOAT | 34 | DXGI_FORMAT_R16G16_FLOAT |
| DX_FORMAT_R16G16_UNORM | 35 | DXGI_FORMAT_R16G16_UNORM |
| DX_FORMAT_R16G16_UINT | 36 | DXGI_FORMAT_R16G16_UINT |
| DX_FORMAT_R16G16_SNORM | 37 | DXGI_FORMAT_R16G16_SNORM |
| DX_FORMAT_R16G16_SINT | 38 | DXGI_FORMAT_R16G16_SINT |
| DX_FORMAT_R32_TYPELESS | 39 | DXGI_FORMAT_R32_TYPELESS |
| DX_FORMAT_D32_FLOAT | 40 | DXGI_FORMAT_D32_FLOAT |
| DX_FORMAT_R32_FLOAT | 41 | DXGI_FORMAT_R32_FLOAT |
| DX_FORMAT_R32_UINT | 42 | DXGI_FORMAT_R32_UINT |
| DX_FORMAT_R32_SINT | 43 | DXGI_FORMAT_R32_SINT |
| DX_FORMAT_R24G8_TYPELESS | 44 | DXGI_FORMAT_R24G8_TYPELESS |
| DX_FORMAT_D24_UNORM_S8_UINT | 45 | DXGI_FORMAT_D24_UNORM_S8_UINT |
| DX_FORMAT_R24_UNORM_X8_TYPELESS | 46 | DXGI_FORMAT_R24_UNORM_X8_TYPELESS |
| DX_FORMAT_X24_TYPELESS_G8_UINT | 47 | DXGI_FORMAT_X24_TYPELESS_G8_UINT |
| DX_FORMAT_R8G8_TYPELESS | 48 | DXGI_FORMAT_R8G8_TYPELESS |
| DX_FORMAT_R8G8_UNORM | 49 | DXGI_FORMAT_R8G8_UNORM |
| DX_FORMAT_R8G8_UINT | 50 | DXGI_FORMAT_R8G8_UINT |
| DX_FORMAT_R8G8_SNORM | 51 | DXGI_FORMAT_R8G8_SNORM |
| DX_FORMAT_R8G8_SINT | 52 | DXGI_FORMAT_R8G8_SINT |
| DX_FORMAT_R16_TYPELESS | 53 | DXGI_FORMAT_R16_TYPELESS |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 匹配 DXGI_FORMAT |
|---|---|---|
| DX_FORMAT_R16_FLOAT | 54 | DXGI_FORMAT_R16_FLOAT |
| DX_FORMAT_D16_UNORM | 55 | DXGI_FORMAT_D16_UNORM |
| DX_FORMAT_R16_UNORM | 56 | DXGI_FORMAT_R16_UNORM |
| DX_FORMAT_R16_UINT | 57 | DXGI_FORMAT_R16_UINT |
| DX_FORMAT_R16_SNORM | 58 | DXGI_FORMAT_R16_SNORM |
| DX_FORMAT_R16_SINT | 59 | DXGI_FORMAT_R16_SINT |
| DX_FORMAT_R8_TYPELESS | 60 | DXGI_FORMAT_R8_TYPELESS |
| DX_FORMAT_R8_UNORM | 61 | DXGI_FORMAT_R8_UNORM |
| DX_FORMAT_R8_UINT | 62 | DXGI_FORMAT_R8_UINT |
| DX_FORMAT_R8_SNORM | 63 | DXGI_FORMAT_R8_SNORM |
| DX_FORMAT_R8_SINT | 64 | DXGI_FORMAT_R8_SINT |
| DX_FORMAT_A8_UNORM | 65 | DXGI_FORMAT_A8_UNORM |
| DX_FORMAT_R1_UNORM | 66 | DXGI_FORMAT_R1_UNORM |
| DX_FORMAT_R9G9B9E5_SHAREDEXP | 67 | DXGI_FORMAT_R9G9B9E5_SHAREDEXP |
| DX_FORMAT_R8G8_B8G8_UNORM | 68 | DXGI_FORMAT_R8G8_B8G8_UNORM |
| DX_FORMAT_G8R8_G8B8_UNORM | 69 | DXGI_FORMAT_G8R8_G8B8_UNORM |
| DX_FORMAT_BC1_TYPELESS | 70 | DXGI_FORMAT_BC1_TYPELESS |
| DX_FORMAT_BC1_UNORM | 71 | DXGI_FORMAT_BC1_UNORM |
| DX_FORMAT_BC1_UNORM_SRGB | 72 | DXGI_FORMAT_BC1_UNORM_SRGB |
| DX_FORMAT_BC2_TYPELESS | 73 | DXGI_FORMAT_BC2_TYPELESS |
| DX_FORMAT_BC2_UNORM | 74 | DXGI_FORMAT_BC2_UNORM |
| DX_FORMAT_BC2_UNORM_SRGB | 75 | DXGI_FORMAT_BC2_UNORM_SRGB |
| DX_FORMAT_BC3_TYPELESS | 76 | DXGI_FORMAT_BC3_TYPELESS |
| DX_FORMAT_BC3_UNORM | 77 | DXGI_FORMAT_BC3_UNORM |
| DX_FORMAT_BC3_UNORM_SRGB | 78 | DXGI_FORMAT_BC3_UNORM_SRGB |
| DX_FORMAT_BC4_TYPELESS | 79 | DXGI_FORMAT_BC4_TYPELESS |
| DX_FORMAT_BC4_UNORM | 80 | DXGI_FORMAT_BC4_UNORM |
| DX_FORMAT_BC4_SNORM | 81 | DXGI_FORMAT_BC4_SNORM |
| DX_FORMAT_BC5_TYPELESS | 82 | DXGI_FORMAT_BC5_TYPELESS |
| DX_FORMAT_BC5_UNORM | 83 | DXGI_FORMAT_BC5_UNORM |
| DX_FORMAT_BC5_SNORM | 84 | DXGI_FORMAT_BC5_SNORM |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 匹配 DXGI_FORMAT |
|---|---|---|
| DX_FORMAT_B5G6R5_UNORM | 85 | DXGI_FORMAT_B5G6R5_UNORM |
| DX_FORMAT_B5G5R5A1_UNORM | 86 | DXGI_FORMAT_B5G5R5A1_UNORM |
| DX_FORMAT_B8G8R8A8_UNORM | 87 | DXGI_FORMAT_B8G8R8A8_UNORM |
| DX_FORMAT_B8G8R8X8_UNORM | 88 | DXGI_FORMAT_B8G8R8X8_UNORM |
| DX_FORMAT_R10G10B10_XR_BIAS_A2_UNORM | 89 | DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM |
| DX_FORMAT_B8G8R8A8_TYPELESS | 90 | DXGI_FORMAT_B8G8R8A8_TYPELESS |
| DX_FORMAT_B8G8R8A8_UNORM_SRGB | 91 | DXGI_FORMAT_B8G8R8A8_UNORM_SRGB |
| DX_FORMAT_B8G8R8X8_TYPELESS | 92 | DXGI_FORMAT_B8G8R8X8_TYPELESS |
| DX_FORMAT_B8G8R8X8_UNORM_SRGB | 93 | DXGI_FORMAT_B8G8R8X8_UNORM_SRGB |
| DX_FORMAT_BC6H_TYPELESS | 94 | DXGI_FORMAT_BC6H_TYPELESS |
| DX_FORMAT_BC6H_UF16 | 95 | DXGI_FORMAT_BC6H_UF16 |
| DX_FORMAT_BC6H_SF16 | 96 | DXGI_FORMAT_BC6H_SF16 |
| DX_FORMAT_BC7_TYPELESS | 97 | DXGI_FORMAT_BC7_TYPELESS |
| DX_FORMAT_BC7_UNORM | 98 | DXGI_FORMAT_BC7_UNORM |
| DX_FORMAT_BC7_UNORM_SRGB | 99 | DXGI_FORMAT_BC7_UNORM_SRGB |
| DX_FORMAT_AYUV | 100 | DXGI_FORMAT_AYUV |
| DX_FORMAT_Y410 | 101 | DXGI_FORMAT_Y410 |
| DX_FORMAT_Y416 | 102 | DXGI_FORMAT_Y416 |
| DX_FORMAT_NV12 | 103 | DXGI_FORMAT_NV12 |
| DX_FORMAT_P010 | 104 | DXGI_FORMAT_P010 |
| DX_FORMAT_P016 | 105 | DXGI_FORMAT_P016 |
| DX_FORMAT_420_OPAQUE | 106 | DXGI_FORMAT_420_OPAQUE |
| DX_FORMAT_YUY2 | 107 | DXGI_FORMAT_YUY2 |
| DX_FORMAT_Y210 | 108 | DXGI_FORMAT_Y210 |
| DX_FORMAT_Y216 | 109 | DXGI_FORMAT_Y216 |
| DX_FORMAT_NV11 | 110 | DXGI_FORMAT_NV11 |
| DX_FORMAT_AI44 | 111 | DXGI_FORMAT_AI44 |
| DX_FORMAT_IA44 | 112 | DXGI_FORMAT_IA44 |
| DX_FORMAT_P8 | 113 | DXGI_FORMAT_P8 |

© 2000-2025, MetaQuotes Ltd.

| ID | 值 | 匹配 DXGI_FORMAT |
|---|---|---|
| DX_FORMAT_A8P8 | 114 | DXGI_FORMAT_A8P8 |
| DX_FORMAT_B4G4R4A4_UNORM | 115 | DXGI_FORMAT_B4G4R4A4_UNORM |
| DX_FORMAT_P208 | 130 | DXGI_FORMAT_P208 |
| DX_FORMAT_V208 | 131 | DXGI_FORMAT_V208 |
| DX_FORMAT_V408 | 132 | DXGI_FORMAT_V408 |
| DX_FORMAT_FORCE_UINT | 0xffffffff | DXGI_FORMAT_FORCE_UINT |

© 2000-2025, MetaQuotes Ltd.

# DXInputCreate

创建着色器输入。

```
int  DXInputCreate(
   int   context,         // 图形环境句柄
   uint  input_size       // 输入大小（以字节为单位）
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*input_size*

[in] 参数结构的大小（以字节为单位）。

返回值

在出现错误时，用于着色器输入或者INVALID_HANDLE的句柄。若要接收错误代码，应该调用GetLastError()函数。

创建的不再使用的句柄应该由DXRelease()函数明确释放。

© 2000-2025, MetaQuotes Ltd.

# DXInputSet

设置着色器输入。

```
bool  DXInputSet(
   int           input,       // 图形环境句柄
   const void&   data         // 设置数据
   );
```

参数

*input*

[in] 在DXInputCreate()中获得的着色器的输入句柄。

*data*

[in] 用于设置着色器输入的数据。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

© 2000-2025, MetaQuotes Ltd.

# DXShaderCreate

创建指定类型的着色器。

```
int  DXShaderCreate(
   int                  context,          // 图形环境句柄
   ENUM_DX_SHADER_TYPE  shader_type,      // 着色器类型
   const string         source,           // 着色器源代码
   const string         entry_point,      // 切入点
   string&              compile_error     // 接收编译器消息的字符串
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*shader_type*

[out] 来自ENUM_DX_SHADER_TYPE枚举的值。

*source*

[in] 在HLSL 5中的着色器源代码。

*entry_point*

[in] 切入点 -源代码中的函数名。

*compile_error*

[in] 接收编译错误的字符串。

返回值

在出现错误时，着色器或者INVALID_HANDLE的句柄。若要接收错误代码，应该调用GetLastError()函数。

注意

创建的不再使用的句柄应该由DXRelease()函数明确释放。

**ENUM_DX_SHADER_TYPE**

| ID | 值 | 描述 |
|---|---|---|
| DX_SHADER_VERTEX | 0 | 顶点着色器 |
| DX_SHADER_GEOMETRY | 1 | 几何着色器 |
| DX_SHADER_PIXEL | 2 | 像素着色器 |

© 2000-2025, MetaQuotes Ltd.

# DXShaderSetLayout

设置顶点着色器的顶点布局。

```
bool  DXShaderSetLayout(
   int                     shader,        // 着色器句柄
   const DXVertexLayout&   layout[]       //
   );
```

参数

*shader*

[in] 在DXShaderCreate()中创建的顶点着色器句柄。

*layout[]*

[in] 顶点字段描述数组。该描述通过DXVertexLayout结构设置：

```
struct DXVertexLayout
  {
   string          semantic_name;        // 在着色器输入签名中与此元素关联的HLSL语义。
   uint            semantic_index;       // 此元素的语义索引。语义索引使用整数索引号修改语义
   ENUM_DX_FORMAT  format;               // 元素数据的数据类型。
  };
```

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

布局应该匹配指定顶点缓冲区中的顶点类型。还应该匹配顶点着色器代码中切入点使用的顶点输入类型。

着色器的顶点缓冲区在DXBufferSet()中设置。

DXVertexLayout结构是D3D11_INPUT_ELEMENT_DESC MSDN结构的一个版本。

© 2000-2025, MetaQuotes Ltd.

# DXShaderInputsSet

设置着色器输入。

```
bool  DXShaderInputsSet(
   int          shader,        // 着色器句柄
   const int&   inputs[]       // 输入句柄数组
   );
```

参数

*shader*

[in] 在DXShaderCreate()中创建的着色器句柄。

*inputs[]*

[in] 使用DXInputCreate()创建的输入句柄数组。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

输入参数的大小应该等于着色器代码中声明的cbuffer对象的数量。

© 2000-2025, MetaQuotes Ltd.

# DXShaderTexturesSet

设置着色器结构。

```
bool  DXShaderTexturesSet(
   int           shader,          // 着色器句柄
   const  int&   textures[]       // 结构句柄数组
   );
```

参数

*shader*

[in] 在DXShaderCreate()中创建的着色器句柄。

*textures[]*

[in] 使用DXTextureCreate()创建的结构句柄数组。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

结构数组的大小应该等于着色器代码中声明的Texture2D对象的数量。

© 2000-2025, MetaQuotes Ltd.

# DXDraw

渲染在DXBufferSet()中设置的顶点缓冲区的顶点

```
bool  DXDraw(
   int   context,                // 图形环境句柄
   uint  start=0,                // 第一个顶点索引
   uint  count=WHOLE_ARRAY       // 顶点数量
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*start*

[in] 渲染的第一个顶点的索引。

*count*

[in] 要渲染的顶点数量。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

应使用DXShaderSet()初步设置着色器来渲染顶点。

© 2000-2025, MetaQuotes Ltd.

# DXDrawIndexed

渲染由DXBufferSet()的索引缓冲区描述的图形原语。

```
bool  DXDrawIndexed(
   int   context,                // 图形环境句柄
   uint  start=0,                // 第一个原语索引
   uint  count=WHOLE_ARRAY       // 原语数量
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*start*

[in] 用于渲染的第一个原语的索引。

*count*

[in] 用于渲染的原语数量。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

使用DXPrimiveTopologySet()设置索引缓冲区描述的原语类型。

初步设置DXBufferSet()中的订单缓冲区来渲染原语。

此外，应使用DXShaderSet()初步设置着色器。

© 2000-2025, MetaQuotes Ltd.

# DXPrimiveTopologySet

设置使用DXDrawIndexed()渲染的原语类型。

```
bool  DXPrimiveTopologySet(
   int                          context,                // 图形环境句柄
   ENUM_DX_PRIMITIVE_TOPOLOGY   primitive_topology      // 原语类型
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*primitive_topology*

[in] 来自ENUM_DX_PRIMITIVE_TOPOLOGY枚举的值。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

ENUM_DX_PRIMITIVE_TOPOLOGY

| ID | 值 | 匹配 D3D11_PRIMITIVE_TOPOLOGY |
|---|---|---|
| DX_PRIMITIVE_TOPOLOGY_POINTLIST | 1 | D3D11_PRIMITIVE_TOPOLOGY_POINTLIST |
| DX_PRIMITIVE_TOPOLOGY_LINELIST | 2 | D3D11_PRIMITIVE_TOPOLOGY_LINELIST |
| DX_PRIMITIVE_TOPOLOGY_LINESTRIP | 3 | D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP |
| DX_PRIMITIVE_TOPOLOGY_TRIANGLELIST | 4 | D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST |
| DX_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP | 5 | D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP |
| DX_PRIMITIVE_TOPOLOGY_LINELIST_ADJ | 6 | D3D11_PRIMITIVE_TOPOLOGY_LINELIST_ADJ |
| DX_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ | 7 | D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ |
| DX_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ | 8 | D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ |
| DX_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ | 9 | D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ |

© 2000-2025, MetaQuotes Ltd.

# DXBufferSet

设置当前渲染的缓冲区。

```
bool  DXBufferSet(
   int   context,                // 图形环境句柄
   int   buffer,                 // 顶点或索引缓冲区句柄
   uint  start=0,                // 初始索引
   uint  count=WHOLE_ARRAY       // 元素数
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*buffer*

[in] 在DXBufferCreate()中创建的顶点或索引缓冲区的句柄。

*start*

[in] 缓冲区第一个元素的索引。默认情况下，使用缓冲区开始的数据。

*count*

[in] 将要使用的值数。默认值是所有缓冲区值。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

应该调用DXBufferSet()函数来设置订单和索引缓冲区，以便使用DXDraw()进行渲染。

© 2000-2025, MetaQuotes Ltd.

# DXShaderSet

设置渲染的着色器。

```
bool  DXShaderSet(
   int  context,       // 图形环境句柄
   int  shader         // 着色器句柄
   );
```

参数

*context*

[in] 在DXContextCreate()中创建的图形环境的句柄。

*shader*

[in] 在DXShaderCreate()中创建的着色器句柄。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

几种着色器类型可以同时用于渲染（顶点着色器、几何着色器和像素着色器）。

© 2000-2025, MetaQuotes Ltd.

# DXHandleType

返回句柄类型。

```
ENUM_DX_HANDLE_TYPE  DXHandleType(
   int  handle      // 句柄
   );
```

参数

*handle*

[in] 句柄。

返回值

来自ENUM_DX_HANDLE_TYPE枚举的值

**ENUM_DX_HANDLE_TYPE**

| ID | 值 | 描述 |
|---|---|---|
| DX_HANDLE_INVALID | 0 | 无效句柄 |
| DX_HANDLE_CONTEXT | 1 | 图形环境句柄 |
| DX_HANDLE_SHADER | 2 | 着色器句柄 |
| DX_HANDLE_BUFFER | 3 | 顶点或索引缓冲区句柄 |
| DX_HANDLE_INPUT | 4 | 着色器输入句柄 |
| DX_HANDLE_TEXTURE | 5 | 结构句柄 |

© 2000-2025, MetaQuotes Ltd.

# DXRelease

释放句柄。

```
bool  DXRelease(
   int  handle      // 句柄
   );
```

参数

*context*

[in] 释放句柄。

返回值

如果执行成功，返回true，否则 - false。若要接收错误代码，应该调用GetLastError()函数。

注意

创建的不再使用的所有句柄应该由DXRelease()函数明确释放。

© 2000-2025, MetaQuotes Ltd.

# MetaTrader与Python集成的模块

MQL5语言专门用于开发金融市场中的高性能交易应用程序，是算法交易使用的其他专门语言中无法比拟的。MQL5程序的语法和速度尽可能接近C++语言，支持OpenCL和与MS VisualStudio的集成。也可使用统计，模糊逻辑和 ALGLIB程序库。MetaEditor开发环境通过"smart"函数导入对.NET程序库提供原生支持，消除开发特殊包装样式的需求。还可以使用第三方C++ DLL。C++源代码文件(CPP和H)可以直接从编辑器中编辑并编译成DLL。安装在用户电脑上的Microsoft Visual Studio也可作此用途。

Python是一种用于开发脚本和应用程序的现代高水平编程语言。它包含用于机器学习、自动化处理以及数据分析和可视化的多个程序库。

MetaTrader Python程序包是为方便快速地从MetaTrader 5程序端直接通过处理器通信获得交易所数据而设计的。通过这种方式接收的数据可以进一步用于统计计算和机器学习。

从命令行安装程序包：

```
pip安装MetaTrader5
```

从命令行更新程序包：

```
pip安装 --更新MetaTrader5
```

用于集成MetaTrader 5和Python的函数

| 函数 | 操作 |
|---|---|
| initialize | 建立与MetaTrader 5程序端的连接 |
| login | 使用指定参数连接交易账户 |
| shutdown | 关于之前建立的与MetaTrader 5程序端的连接 |
| version | 返回MetaTrader 5程序端版本 |
| last_error | 返回最后一个错误的数据 |
| account_info | 获取当前交易账户的信息 |
| terminal_Info | 获取所连接的MetaTrader 5程序端的状态和参数 |
| symbols_total | 获取MetaTrader 5程序端中所有交易品种的数量 |
| symbols_get | 获取MetaTrader 5程序端中的所有交易品种 |
| symbol_info | 获取指定交易品种的数据 |
| symbol_info_tick | 获取指定交易品种的最后报价 |
| symbol_select | 在MarketWatch（市场报价）窗口中选择一个交易品种或从该窗口移除一个交易品种 |
| market_book_add | 订阅MetaTrader 5程序端的指定交易品种的市场深度更改事件 |
| market_book_get | 从BookInfo返回一个元组，其中包含指定交易品种的市场深度条目 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 操作 |
| --- | --- |
| market_book_release | 取消订阅MetaTrader 5程序端的指定交易品种的市场深度更改事件 |
| copy_rates_from | 从指定日期开始从MetaTrader 5程序端获取柱形图 |
| copy_rates_from_pos | 从指定索引开始从MetaTrader 5程序端获取柱形图 |
| copyrates_range | 从MetaTrader 5程序端获取指定日期范围内的柱形图 |
| copy_ticks_from | 从指定日期开始从MetaTrader 5程序端获取报价 |
| copy_ticks_range | 从MetaTrader 5程序端获取指定日期范围内的报价 |
| orders_total | 获取活动订单的数量。 |
| orders_get | 获取可通过交易品种或单号筛选的活动订单 |
| order_calc_margin | 返回预付款（用账户货币表示）来执行指定的交易操作 |
| order_calc_profit | 返回指定交易操作的盈利（用账户货币表示） |
| order_check | 检查是否有足够的资金执行所需的交易操作 |
| order_send | 发送请求来执行交易操作。 |
| positions_total | 获取未结持仓的数量 |
| positions_get | 获取可通过交易品种或单号筛选的未结持仓 |
| history_orders_total | 获取交易历史中指定时间间隔内的订单数量 |
| history_orders_get | 获取可通过单号或持仓筛选的交易历史中的订单 |
| history_deals_total | 获取交易历史中指定时间间隔内的交易数量 |
| history_deals_get | 获取可通过单号或持仓筛选的交易历史中的交易 |

## 连接Python与MetaTrader 5的示例

1. 从https://www.python.org/downloads/windows下载最新版Python 3.8
2. 安装Python时，请检查“将Python 3.8添加到PATH%”，以便能够从命令行运行Python脚本。
3. 从命令行安装MetaTrader 5模块

```
pip安装MetaTrader5
```

4. 添加matplotlib和pandas程序包

```
pip安装matplotlib
pip安装pandas
```

5. 启动测试脚本

```
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
import MetaTrader5 as mt5
```

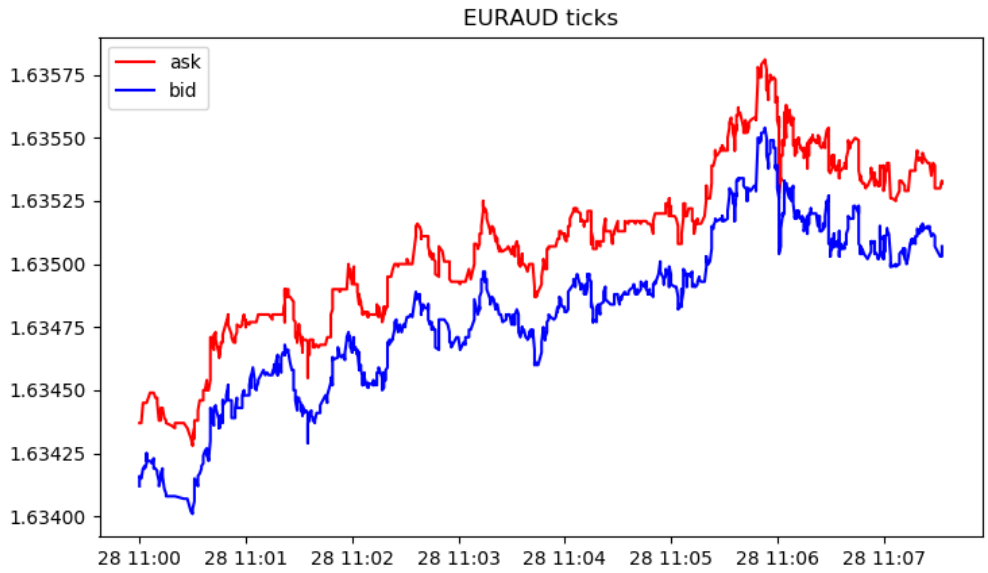
© 2000-2025, MetaQuotes Ltd.

```
# 连接到MetaTrader 5
if not mt5.initialize():
    print("initialize() failed")
    mt5.shutdown()

# 请求连接状态和参数
print(mt5.terminal_info())
# 获取有关MetaTrader 5版本的数据
print(mt5.version())

# 从EURAUD请求1000个报价
euraud_ticks = mt5.copy_ticks_from("EURAUD", datetime(2020,1,28,13), 1000, mt5.COPY_TI
# 请求2019.04.01 13:00 - 2019.04.02 13:00之间的AUDUSD报价
audusd_ticks = mt5.copy_ticks_range("AUDUSD", datetime(2020,1,27,13), datetime(2020,1,

# 通过多种方式获取不同交易品种的柱形图
eurusd_rates = mt5.copy_rates_from("EURUSD", mt5.TIMEFRAME_M1, datetime(2020,1,28,13),
eurgbp_rates = mt5.copy_rates_from_pos("EURGBP", mt5.TIMEFRAME_M1, 0, 1000)
eurcad_rates = mt5.copy_rates_range("EURCAD", mt5.TIMEFRAME_M1, datetime(2020,1,27,13)

# 断开与MetaTrader 5的连接
mt5.shutdown()

#DATA
print('euraud_ticks(', len(euraud_ticks), ')')
for val in euraud_ticks[:10]: print(val)

print('audusd_ticks(', len(audusd_ticks), ')')
for val in audusd_ticks[:10]: print(val)

print('eurusd_rates(', len(eurusd_rates), ')')
for val in eurusd_rates[:10]: print(val)

print('eurgbp_rates(', len(eurgbp_rates), ')')
for val in eurgbp_rates[:10]: print(val)

print('eurcad_rates(', len(eurcad_rates), ')')
for val in eurcad_rates[:10]: print(val)

#PLOT
# 从所获得的数据创建DataFrame
ticks_frame = pd.DataFrame(euraud_ticks)
# 将时间（以秒为单位）转换为日期时间格式
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')
# 在图表上显示报价
plt.plot(ticks_frame['time'], ticks_frame['ask'], 'r-', label='ask')
plt.plot(ticks_frame['time'], ticks_frame['bid'], 'b-', label='bid')
```

© 2000-2025, MetaQuotes Ltd.

```
# 显示图例
plt.legend(loc='upper left')

# 添加标题
plt.title('EURAUD ticks')

# 显示图表
plt.show()
```

6. 获取数据和图表




```
[2, 'MetaQuotes-Demo', '16167573']
[500, 2325, '19 Feb 2020']

euraud_ticks( 1000 )
(1580209200, 1.63412, 1.63437, 0., 0, 1580209200067, 130, 0.)
(1580209200, 1.63416, 1.63437, 0., 0, 1580209200785, 130, 0.)
(1580209201, 1.63415, 1.63437, 0., 0, 1580209201980, 130, 0.)
(1580209202, 1.63419, 1.63445, 0., 0, 1580209202192, 134, 0.)
(1580209203, 1.6342, 1.63445, 0., 0, 1580209203004, 130, 0.)
(1580209203, 1.63419, 1.63445, 0., 0, 1580209203487, 130, 0.)
(1580209203, 1.6342, 1.63445, 0., 0, 1580209203694, 130, 0.)
(1580209203, 1.63419, 1.63445, 0., 0, 1580209203990, 130, 0.)
(1580209204, 1.63421, 1.63445, 0., 0, 1580209204194, 130, 0.)
(1580209204, 1.63425, 1.63445, 0., 0, 1580209204392, 130, 0.)
audusd_ticks( 40449 )
(1580122800, 0.67858, 0.67868, 0., 0, 1580122800244, 130, 0.)
(1580122800, 0.67858, 0.67867, 0., 0, 1580122800429, 4, 0.)
(1580122800, 0.67858, 0.67865, 0., 0, 1580122800817, 4, 0.)
(1580122801, 0.67858, 0.67866, 0., 0, 1580122801618, 4, 0.)
(1580122802, 0.67858, 0.67865, 0., 0, 1580122802928, 4, 0.)
(1580122809, 0.67855, 0.67865, 0., 0, 1580122809526, 130, 0.)
(1580122809, 0.67855, 0.67864, 0., 0, 1580122809699, 4, 0.)
```

© 2000-2025, MetaQuotes Ltd.

```
(1580122813, 0.67855, 0.67863, 0., 0, 1580122813576, 4, 0.)
(1580122815, 0.67856, 0.67863, 0., 0, 1580122815190, 130, 0.)
(1580122815, 0.67855, 0.67863, 0., 0, 1580122815479, 130, 0.)
eurusd_rates( 1000 )
(1580149260, 1.10132, 1.10151, 1.10131, 1.10149, 44, 1, 0)
(1580149320, 1.10149, 1.10161, 1.10143, 1.10154, 42, 1, 0)
(1580149380, 1.10154, 1.10176, 1.10154, 1.10174, 40, 2, 0)
(1580149440, 1.10174, 1.10189, 1.10168, 1.10187, 47, 1, 0)
(1580149500, 1.10185, 1.10191, 1.1018, 1.10182, 53, 1, 0)
(1580149560, 1.10182, 1.10184, 1.10176, 1.10183, 25, 3, 0)
(1580149620, 1.10183, 1.10187, 1.10177, 1.10187, 49, 2, 0)
(1580149680, 1.10187, 1.1019, 1.1018, 1.10187, 53, 1, 0)
(1580149740, 1.10187, 1.10202, 1.10187, 1.10198, 28, 2, 0)
(1580149800, 1.10198, 1.10198, 1.10183, 1.10188, 39, 2, 0)
eurgbp_rates( 1000 )
(1582236360, 0.83767, 0.83767, 0.83764, 0.83765, 23, 9, 0)
(1582236420, 0.83765, 0.83765, 0.83764, 0.83765, 15, 8, 0)
(1582236480, 0.83765, 0.83766, 0.83762, 0.83765, 19, 7, 0)
(1582236540, 0.83765, 0.83768, 0.83758, 0.83763, 39, 6, 0)
(1582236600, 0.83763, 0.83768, 0.83763, 0.83767, 21, 6, 0)
(1582236660, 0.83767, 0.83775, 0.83765, 0.83769, 63, 5, 0)
(1582236720, 0.83769, 0.8377, 0.83758, 0.83764, 40, 7, 0)
(1582236780, 0.83766, 0.83769, 0.8376, 0.83766, 37, 6, 0)
(1582236840, 0.83766, 0.83772, 0.83763, 0.83772, 22, 6, 0)
(1582236900, 0.83772, 0.83773, 0.83768, 0.8377, 36, 5, 0)
eurcad_rates( 1441 )
(1580122800, 1.45321, 1.45329, 1.4526, 1.4528, 146, 15, 0)
(1580122860, 1.4528, 1.45315, 1.45274, 1.45301, 93, 15, 0)
(1580122920, 1.453, 1.45304, 1.45264, 1.45264, 82, 15, 0)
(1580122980, 1.45263, 1.45279, 1.45231, 1.45277, 109, 15, 0)
(1580123040, 1.45275, 1.4528, 1.45259, 1.45271, 53, 14, 0)
(1580123100, 1.45273, 1.45285, 1.45269, 1.4528, 62, 16, 0)
(1580123160, 1.4528, 1.45284, 1.45267, 1.45282, 64, 14, 0)
(1580123220, 1.45282, 1.45299, 1.45261, 1.45272, 48, 14, 0)
(1580123280, 1.45272, 1.45275, 1.45255, 1.45275, 74, 14, 0)
(1580123340, 1.45275, 1.4528, 1.4526, 1.4528, 94, 13, 0)
```

© 2000-2025, MetaQuotes Ltd.

# initialize

建立与MetaTrader 5程序端的连接。共有三个调用选项。

无参数调用。自动找到程序端连接。

```
initialize()
```

调用指定要连接到MetaTrader 5程序端的路径。

```
initialize(
   path                          // 到MetaTrader 5程序端EXE文件的路径
   )
```

调用指定交易账户路径和参数。

```
initialize(
   path                          // 到MetaTrader 5程序端EXE文件的路径
   timeout=TIMEOUT,              // 超时
   login=LOGIN,                  // 账号
   password="PASSWORD",          // 密码
   server="SERVER"               // 在程序端指定的服务器名称
   )
```

参数

*path*

[in] metatrader.exe或metatrader64.exe文件的路径。可选的未命名参数。首先显示，没有参数名称。如果没有指定此路径，模块会尝试自己查找可执行文件。

*timeout=TIMEOUT*

[in] 连接超时，以毫秒计算。可选的命名参数。如果没有指定，则应用60 000 (60秒)的值。如果没有在指定时间内建立连接，则调用强制终止并生成异常。

*login=LOGIN*

[in] 交易账号。可选的命名参数。如果没有指定，则使用最后一个交易账户。

*password="PASSWORD"*

[in] 交易账号密码。可选的命名参数。如果没有设置密码，则自动应用在程序端数据库中保存的密码。

*server="SERVER"*

[in] 交易服务器名称。可选的命名参数。如果没有设置服务器，则自动应用最后使用的服务器。

返回值

成功连接MetaTrader 5程序端，返回True，否则，返回False。

注意

如果有需要，当执行initialize()调用时，启动MetaTrader 5程序端建立连接。

例如：

© 2000-2025, MetaQuotes Ltd.

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立MetaTrader 5到指定交易账户的连接
if not mt5.initialize(login=25115284, server="MetaQuotes-Demo",password="4zatlbqx"):
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示有关连接状态、服务器名称和交易账户的数据
print(mt5.terminal_info())
# 显示有关MetaTrader 5版本的数据
print(mt5.version())

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另 见

shutdown、terminal_info、version

© 2000-2025, MetaQuotes Ltd.

# login

使用指定参数连接交易账户。

```
login(
   login,                    // 账号
   password="PASSWORD",      // 密码
   server="SERVER",          // 在程序端指定的服务器名称
   timeout=TIMEOUT           // 超时
   )
```

参数

*login*

[in] 交易账号。所需的未命名参数。

*password*

[in] 交易账号密码。可选的命名参数。如果没有设置密码，则自动应用在程序端数据库中保存的密码。

*server*

[in] 交易服务器名称。可选的命名参数。如果没有设置服务器，则自动应用最后使用的服务器。

*timeout=TIMEOUT*

[in] 连接超时，以毫秒计算。可选的命名参数。如果没有指定，则应用60 000 (60秒)的值。如果没有在指定时间内建立连接，则调用强制终止并生成异常。

返回值

成功连接交易账户，返回True，否则，返回False。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示有关MetaTrader 5版本的数据
print(mt5.version())
# 连接到指定密码和服务器的交易账户
account=17221085
authorized=mt5.login(account)  # the terminal database password is applied if connecti
if authorized:
    print("connected to account #{}".format(account))
其他:
```

© 2000-2025, MetaQuotes Ltd.

```
    print("failed to connect at account #{}, error code: {}".format(account, mt5.last_

# 现在连接到指定密码的另一个交易账户
account=25115284
authorized=mt5.login(account, password="gqrtz0lbdm")
if authorized:
    # display trading account data 'as is'
    print(mt5.account_info())
    # display trading account data in the form of a list
    print("Show account_info()._asdict():")
    account_info_dict = mt5.account_info()._asdict()
    for prop in account_info_dict:
        print("  {}={}".format(prop, account_info_dict[prop]))
其他:
    print("failed to connect at account #{}, error code: {}".format(account, mt5.last_

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()


Result:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
[500, 2367, '23 Mar 2020']

connected to account #17221085

connected to account #25115284
AccountInfo(login=25115284, trade_mode=0, leverage=100, limit_orders=200, margin_so_mo
account properties:
   login=25115284
   trade_mode=0
   leverage=100
   limit_orders=200
   margin_so_mode=0
   trade_allowed=True
   trade_expert=True
   margin_mode=2
   currency_digits=2
   fifo_close=False
   balance=99588.33
   credit=0.0
   profit=-45.23
   equity=99543.1
   margin=54.37
   margin_free=99488.73
   margin_level=183084.6054809638
   margin_so_call=50.0
   margin_so_so=30.0
```

© 2000-2025, MetaQuotes Ltd.

```
    margin_initial=0.0
    margin_maintenance=0.0
    assets=0.0
    liabilities=0.0
    commission_blocked=0.0
    name=James Smith
    server=MetaQuotes-Demo
    currency=USD
    company=MetaQuotes Software Corp.
```

另见

initialize、shutdown

© 2000-2025, MetaQuotes Ltd.

# shutdown

关于之前建立的与MetaTrader 5程序端的连接。

```
shutdown()
```

返回值

无。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed")
    quit()

# 显示有关连接状态、服务器名称和交易账户的数据
print(mt5.terminal_info())
# 显示有关MetaTrader 5版本的数据
print(mt5.version())

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另见

initialize、login_py、terminal_info、version

© 2000-2025, MetaQuotes Ltd.

# version

返回MetaTrader5程序端版本。

```
version()
```

返回值

返回MetaTrader5程序端版本、build和发布日期。错误情况下返回None。可使用last_error()获取错误信息。

注意

version()函数以三个值元组的形式返回程序端版本、build和发布日期：

| 类型 | 描述 | 样本值 |
| --- | --- | --- |
| 整数 | MetaTrader5程序端版本 | 500 |
| 整数 | Build | 2007 |
| 字符串 | Build发布日期 | '25 Feb 2019' |

例如：

```
import MetaTrader5 as mt5
import pandas as pd
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示有关MetaTrader 5版本的数据
print(mt5.version())

# display data on connection status, server name and trading account 'as is'
print(mt5.terminal_info())
print()

# 获取词典形式的属性
terminal_info_dict=mt5.terminal_info()._asdict()
# 将词典转换为DataFrame和print
df=pd.DataFrame(list(terminal_info_dict.items()),columns=['property','value'])
print("terminal_info() as dataframe:")
print(df[:-1])

# 断开与MetaTrader 5程序端的连接
```

© 2000-2025, MetaQuotes Ltd.

```
mt5.shutdown()


Result:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
[500, 2367, '23 Mar 2020']
TerminalInfo(community_account=True, community_connection=True, connected=True, dlls_a

terminal_info() as dataframe:
                  property                          value
0       community_account                           True
1    community_connection                           True
2               connected                           True
3            dlls_allowed                          False
4           trade_allowed                          False
5       tradeapi_disabled                          False
6           email_enabled                          False
7             ftp_enabled                          False
8   notifications_enabled                          False
9                    mqid                          False
10                  build                           2367
11                maxbars                           5000
12               codepage                           1251
13             ping_last                          77881
14      community_balance                        707.107
15         retransmission                              0
16                company     MetaQuotes Software Corp.
17                   name                   MetaTrader 5
18               language                        Russian
19                   path  E:\ProgramFiles\MetaTrader 5
20              data_path  E:\ProgramFiles\MetaTrader 5
```

另见

initialize、shutdown、terminal_info

© 2000-2025, MetaQuotes Ltd.

# last_error

返回最后一个错误的数据。

```
last_error()
```

返回值

以元组形式返回最后一个错误代码和描述。

注意

last_error()允许在MetaTrader 5程序库函数执行失败的情况下获取错误代码。它类似于GetLastError()。然而，它应用自己的错误代码。可能值：

| 常量 | 值 | 描述 |
| --- | --- | --- |
| RES_S_OK | 1 | 通用成功 |
| RES_E_FAIL | -1 | 通用失败 |
| RES_E_INVALID_PARAMS | -2 | 无效参数 |
| RES_E_NO_MEMORY | -3 | 无存储条件 |
| RES_E_NOT_FOUND | -4 | 没有历史 |
| RES_E_INVALID_VERSION | -5 | 无效版本 |
| RES_E_AUTH_FAILED | -6 | 授权失败 |
| RES_E_UNSUPPORTED | -7 | 不受支持的方法 |
| RES_E_AUTO_TRADING_DISABLED | -8 | 禁用自动交易 |
| RES_E_INTERNAL_FAIL | -10000 | 内部IPC一般错误 |
| RES_E_INTERNAL_FAIL_SEND | -10001 | 内部IPC发送失败 |
| RES_E_INTERNAL_FAIL_RECEIVE | -10002 | 内部IPC接收失败 |
| RES_E_INTERNAL_FAIL_INIT | -10003 | 内部IPC初始化失败 |
| RES_E_INTERNAL_FAIL_CONNECT | -10003 | 内部IPC没有ipc |
| RES_E_INTERNAL_FAIL_TIMEOUT | -10005 | 内部超时 |

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()
```

© 2000-2025, MetaQuotes Ltd.

```
# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另 见

version、GetLastError

© 2000-2025, MetaQuotes Ltd.

# account_info

获取当前交易账户的信息。

```
account_info()
```

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数返回在一次调用中使用AccountInfoInteger、AccountInfoDouble和AccountInfoString可获得的所有数据。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 连接到指定密码和服务器的交易账户
authorized=mt5.login(25115284, password="gqz0343lbdm")
if authorized:
    account_info=mt5.account_info()
    if account_info!=None:
        # display trading account data 'as is'
        print(account_info)
        # display trading account data in the form of a dictionary
        print("Show account_info()._asdict():")
        account_info_dict = mt5.account_info()._asdict()
        for prop in account_info_dict:
            print("  {}={}".format(prop, account_info_dict[prop]))
        print()

       # 将词典转换为DataFrame和print
        df=pd.DataFrame(list(account_info_dict.items()),columns=['property','value'])
        print("account_info() as dataframe:")
        print(df)
其他:
    print("failed to connect to trade account 25115284 with password=gqz0343lbdm, erro

# 断开与MetaTrader 5程序端的连接
```

© 2000-2025, MetaQuotes Ltd.

```
mt5.shutdown()


结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
AccountInfo(login=25115284, trade_mode=0, leverage=100, limit_orders=200, margin_so_mo
Show account_info()._asdict():
  login=25115284
  trade_mode=0
  leverage=100
  limit_orders=200
  margin_so_mode=0
  trade_allowed=True
  trade_expert=True
  margin_mode=2
  currency_digits=2
  fifo_close=False
  balance=99511.4
  credit=0.0
  profit=41.82
  equity=99553.22
  margin=98.18
  margin_free=99455.04
  margin_level=101398.67590140559
  margin_so_call=50.0
  margin_so_so=30.0
  margin_initial=0.0
  margin_maintenance=0.0
  assets=0.0
  liabilities=0.0
  commission_blocked=0.0
  server=MetaQuotes-Demo
  currency=USD
  company=MetaQuotes Software Corp.

account_info() as dataframe
              property                     value
0                 login                  25115284
1            trade_mode                         0
2              leverage                       100
3          limit_orders                       200
4       margin_so_mode                         0
5        trade_allowed                      True
6         trade_expert                      True
7          margin_mode                         2
8      currency_digits                         2
9           fifo_close                     False
10             balance                   99588.3
```

© 2000-2025, MetaQuotes Ltd.

```
11              credit                          0
12              profit                     -45.13
13              equity                    99543.2
14              margin                      54.37
15         margin_free                    99488.8
16        margin_level                     183085
17      margin_so_call                         50
18        margin_so_so                         30
19      margin_initial                          0
20  margin_maintenance                          0
21              assets                          0
22         liabilities                          0
23  commission_blocked                          0
24                name                 James Smith
25              server             MetaQuotes-Demo
26            currency                        USD
27             company  MetaQuotes Software Corp.
```

另 见

initialize、shutdown、login

© 2000-2025, MetaQuotes Ltd.

# terminal_info

获取已连接的MetaTrader 5客户端的状态和设置。

```
terminal_info()
```

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数返回在一次调用中使用TerminalInfoInteger、 TerminalInfoDouble和TerminalInfoDouble可获得的所有数据。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示有关MetaTrader 5版本的数据
print(mt5.version())
# 显示有关程序端设置和状态的信息
terminal_info=mt5.terminal_info()
if terminal_info!=None:
    # display the terminal data 'as is'
    print(terminal_info)
    # display data in the form of a list
    print("Show terminal_info()._asdict():")
    terminal_info_dict = mt5.terminal_info()._asdict()
    for prop in terminal_info_dict:
        print("  {}={}".format(prop, terminal_info_dict[prop]))
    print()
   # 将词典转换为DataFrame和print
    df=pd.DataFrame(list(terminal_info_dict.items()),columns=['property','value'])
    print("terminal_info() as dataframe:")
    print(df)

# 断开与MetaTrader 5程序端的连接
```

© 2000-2025, MetaQuotes Ltd.

```
mt5.shutdown()


结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
[500, 2366, '20 Mar 2020']
TerminalInfo(community_account=True, community_connection=True, connected=True,....
Show terminal_info()._asdict():
  community_account=True
  community_connection=True
  connected=True
  dlls_allowed=False
  trade_allowed=False
  tradeapi_disabled=False
  email_enabled=False
  ftp_enabled=False
  notifications_enabled=False
  mqid=False
  build=2366
  maxbars=5000
  codepage=1251
  ping_last=77850
  community_balance=707.10668201585
  retransmission=0.0
  company=MetaQuotes Software Corp.
  name=MetaTrader 5
  language=Russian
  path=E:\ProgramFiles\MetaTrader 5
  data_path=E:\ProgramFiles\MetaTrader 5
  commondata_path=C:\Users\Rosh\AppData\Roaming\MetaQuotes\Terminal\Common

terminal_info() as dataframe:
                  property                      value
0       community_account                       True
1    community_connection                       True
2               connected                       True
3            dlls_allowed                      False
4           trade_allowed                      False
5       tradeapi_disabled                      False
6           email_enabled                      False
7             ftp_enabled                      False
8   notifications_enabled                      False
9                    mqid                      False
10                  build                       2367
11                maxbars                       5000
12               codepage                       1251
13             ping_last                      80953
14      community_balance                    707.107
```

© 2000-2025, MetaQuotes Ltd.

```
15         retransmission                   0.063593
16                company  MetaQuotes Software Corp.
17                   name                MetaTrader 5
18               language                     Russian
```

另见

initialize、shutdown、version

© 2000-2025, MetaQuotes Ltd.

# symbols_total

获取MetaTrader5程序端中所有交易品种的数量。

```
symbols_total()
```

返回值

整数值。

注意

该函数类似于SymbolsTotal()。然而，它返回所有交易品种的数量，包括自定义交易品种，也包括市场报价中禁用的交易品种。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取交易品种的数量
symbols=mt5.symbols_total()
if symbols>0:
    print("Total symbols =",symbols)
其他:
    print("symbols not found")

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另见

symbols_get、symbol_select、symbol_info

© 2000-2025, MetaQuotes Ltd.

# symbols_get

获取MetaTrader5程序端中的所有交易品种。

```
symbols_get(
   group="GROUP"       // 交易品种选择过滤器
)
```

*group="GROUP"*

[in] 用于跑了一组必要交易品种的过滤器。可选参数。如果指定了组，则函数只返回满足指定条件的交易品种。

返回值

以元组的形式返回交易品种。错误情况下返回None。可使用last_error()获取错误信息。

注意

*group*参数可以根据名称对交易品种进行分类。'*'可以用在字符串的开头和结尾。

*group*参数可被用作命名的参数或未命名的参数。两个选项工作方式相同。命名的选项(group="GROUP")使代码更易于阅读。

*group*参数可以包含多个逗号分隔条件。条件可以使用'*'设置为掩码。逻辑否定字符'!'可以用来表示排除条件。所有条件依序应用，这意味着应该先指定包含到组中的条件，然后再指定排除条件。例如，*group*="*, !EUR"意味着应该先选择所有交易品种，然后排除名称中包含"EUR"的交易品种。

不同于symbol_info(), symbols_get()函数在单个调用中返回所有请求交易品种的数据。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取所有交易品种
symbols=mt5.symbols_get()
print('Symbols: ', len(symbols))
count=0
# 显示前五个交易品种
for s in symbols:
    count+=1
    print("{}. {}".format(count,s.name))
    if count==5: break
print()
```

© 2000-2025, MetaQuotes Ltd.

```
# 获取名称中包含RU的交易品种
ru_symbols=mt5.symbols_get("*RU*")
print('len(*RU*): ', len(ru_symbols))
for s in ru_symbols:
    print(s.name)
print()

# 获取名称中不包含USD、EUR、JPY和GBP的交易品种
group_symbols=mt5.symbols_get(group="*,!*USD*,!*EUR*,!*JPY*,!*GBP*")
print('len(*,!*USD*,!*EUR*,!*JPY*,!*GBP*):', len(group_symbols))
for s in group_symbols:
    print(s.name,":",s)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()

结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
交易品种:84
1. EURUSD
2. GBPUSD
3. USDCHF
4. USDJPY
5. USDCNH

len(*RU*):  8
EURUSD
USDRUB
USDRUR
EURRUR
EURRUB
FORTS.RUB.M5
EURUSD_T20
EURUSD4

len(*,!*USD*,!*EUR*,!*JPY*,!*GBP*):  13
AUDCAD : SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_dea
AUDCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
AUDNZD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
CADCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
NZDCAD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
NZDCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
NZDSGD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
CADMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
CHFMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
NZDMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_d
FORTS.RTS.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
FORTS.RUB.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
```

© 2000-2025, MetaQuotes Ltd.

```
FOREX.CHF.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
```

另见

symbols_total、symbol_select、symbol_info

© 2000-2025, MetaQuotes Ltd.

# symbol_info

获取指定交易品种的数据。

```
symbol_info(
   symbol      // 交易品种名称
)
```

*symbol*

[in] 交易品种名称。所需的未命名参数。

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数返回在一次调用中使用SymbolInfoInteger、SymbolInfoDouble和SymbolInfoString可获得的所有数据。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 尝试在市场报价中启用显示EURJPY交易品种
selected=mt5.symbol_select("EURJPY",True)
if not selected:
    print("Failed to select EURJPY")
    mt5.shutdown()
    quit()

# 显示EURJPY交易品种属性
symbol_info=mt5.symbol_info("EURJPY")
if symbol_info!=None:
    # display the terminal data 'as is'
    print(symbol_info)
    print("EURJPY: spread =",symbol_info.spread,"  digits =",symbol_info.digits)
    # display symbol properties as a list
    print("Show symbol_info(\"EURJPY\")._asdict():")
    symbol_info_dict = mt5.symbol_info("EURJPY")._asdict()
    for prop in symbol_info_dict:
        print("  {}={}".format(prop, symbol_info_dict[prop]))
```

© 2000-2025, MetaQuotes Ltd.

```
# 断开与MetaTrader 5程序端的连接
mt5.shutdown()


结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, ses
EURJPY: spread = 17   digits = 3
Show symbol_info()._asdict():
  custom=False
  chart_mode=0
  select=True
  visible=True
  session_deals=0
  session_buy_orders=0
  session_sell_orders=0
  volume=0
  volumehigh=0
  volumelow=0
  time=1585069682
  digits=3
  spread=17
  spread_float=True
  ticks_bookdepth=10
  trade_calc_mode=0
  trade_mode=4
  start_time=0
  expiration_time=0
  trade_stops_level=0
  trade_freeze_level=0
  trade_exemode=1
  swap_mode=1
  swap_rollover3days=3
  margin_hedged_use_leg=False
  expiration_mode=7
  filling_mode=1
  order_mode=127
  order_gtc_mode=0
  option_mode=0
  option_right=0
  bid=120.024
  bidhigh=120.506
  bidlow=118.798
  ask=120.041
  askhigh=120.526
  asklow=118.828
  last=0.0
```

© 2000-2025, MetaQuotes Ltd.

```
lasthigh=0.0
lastlow=0.0
volume_real=0.0
volumehigh_real=0.0
volumelow_real=0.0
option_strike=0.0
point=0.001
trade_tick_value=0.8977708350166538
trade_tick_value_profit=0.8977708350166538
trade_tick_value_loss=0.8978272580355541
trade_tick_size=0.001
trade_contract_size=100000.0
trade_accrued_interest=0.0
trade_face_value=0.0
trade_liquidity_rate=0.0
volume_min=0.01
volume_max=500.0
volume_step=0.01
volume_limit=0.0
swap_long=-0.2
swap_short=-1.2
margin_initial=0.0
margin_maintenance=0.0
session_volume=0.0
session_turnover=0.0
session_interest=0.0
session_buy_orders_volume=0.0
session_sell_orders_volume=0.0
session_open=0.0
session_close=0.0
session_aw=0.0
session_price_settlement=0.0
session_price_limit_min=0.0
session_price_limit_max=0.0
margin_hedged=100000.0
price_change=0.0
price_volatility=0.0
price_theoretical=0.0
price_greeks_delta=0.0
price_greeks_theta=0.0
price_greeks_gamma=0.0
price_greeks_vega=0.0
price_greeks_rho=0.0
price_greeks_omega=0.0
price_sensitivity=0.0
basis=
category=
currency_base=EUR
currency_profit=JPY
```

© 2000-2025, MetaQuotes Ltd.

```
currency_margin=EUR
bank=
description=Euro vs Japanese Yen
exchange=
formula=
isin=
name=EURJPY
page=http://www.google.com/finance?q=EURJPY
path=Forex\EURJPY
```

另见

account_info、terminal_info

© 2000-2025, MetaQuotes Ltd.

# symbol_info_tick

获取指定交易品种的最后报价。

```
symbol_info_tick(
   symbol      // 交易品种名称
)
```

*symbol*

[in] 交易品种名称。所需的未命名参数。

返回值

以元组的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数类似于SymbolInfoTick。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 尝试在市场报价中启用显示GBPUSD
selected=mt5.symbol_select("GBPUSD",True)
if not selected:
    print("Failed to select GBPUSD")
    mt5.shutdown()
    quit()

# 显示GBPUSD最后报价
lasttick=mt5.symbol_info_tick("GBPUSD")
print(lasttick)
# 以列表的形式显示报价字段值
print("Show symbol_info_tick(\"GBPUSD\")._asdict():")
symbol_info_tick_dict = mt5.symbol_info_tick("GBPUSD")._asdict()
for prop in symbol_info_tick_dict:
    print("  {}={}".format(prop, symbol_info_tick_dict[prop]))

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

© 2000-2025, MetaQuotes Ltd.

```
结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
Tick(time=1585070338, bid=1.17264, ask=1.17279, last=0.0, volume=0, time_msc=158507033
Show symbol_info_tick._asdict():
  time=1585070338
  bid=1.17264
  ask=1.17279
  last=0.0
  volume=0
  time_msc=1585070338728
  flags=2
  volume_real=0.0
```

另 见

symbol_info、symbol_info

© 2000-2025, MetaQuotes Ltd.

# symbol_select

在MarketWatch（市场报价）窗口中选择一个交易品种或从该窗口移除一个交易品种。

```
symbol_select(
   symbol,       // 交易品种名称
   enable=None   // 启用或禁用
)
```

*symbol*

[in] 交易品种名称。所需的未命名参数。

*enable*

[in] 切换。可选的未命名参数。如果是'false'，则交易品种应从MarketWatch（市场报价）窗口移除。否则，应该在市场报价窗口中将其选定。如果已存在一个交易品种的打开图表或有其未结持仓，则不可以删除此交易品种。

返回值

如果成功返回true，否则 false。

注意

该函数类似于SymbolSelect。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# 建立与MetaTrader 5程序端的连接
if not mt5.initialize(login=25115284, server="MetaQuotes-Demo",password="4zatlbqx"):
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 尝试在市场报价中启用显示EURCAD
selected=mt5.symbol_select("EURCAD",True)
if not selected:
    print("Failed to select EURCAD, error code =",mt5.last_error())
其他:
    symbol_info=mt5.symbol_info("EURCAD")
    print(symbol_info)
    print("EURCAD: currency_base =",symbol_info.currency_base,"  currency_profit =",sy
    print()

    # get symbol properties in the form of a dictionary
    print("Show symbol_info()._asdict():")
    symbol_info_dict = symbol_info._asdict()
    for prop in symbol_info_dict:
```

© 2000-2025, MetaQuotes Ltd.

```
        print("  {}={}".format(prop, symbol_info_dict[prop]))
    print()

   # 将词典转换为DataFrame和print
    df=pd.DataFrame(list(symbol_info_dict.items()),columns=['property','value'])
    print("symbol_info_dict() as dataframe:")
    print(df)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()


结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, ses
EURCAD: currency_base = EUR   currency_profit = CAD   currency_margin = EUR

Show symbol_info()._asdict():
  custom=False
  chart_mode=0
  select=True
  visible=True
  session_deals=0
  session_buy_orders=0
  session_sell_orders=0
  volume=0
  volumehigh=0
  volumelow=0
  time=1585217595
  digits=5
  spread=39
  spread_float=True
  ticks_bookdepth=10
  trade_calc_mode=0
  trade_mode=4
  start_time=0
  expiration_time=0
  trade_stops_level=0
  trade_freeze_level=0
  trade_exemode=1
  swap_mode=1
  swap_rollover3days=3
  margin_hedged_use_leg=False
  expiration_mode=7
  filling_mode=1
  order_mode=127
  order_gtc_mode=0
  option_mode=0
```

© 2000-2025, MetaQuotes Ltd.

```
option_right=0
bid=1.55192
bidhigh=1.55842
bidlow=1.5419800000000001
ask=1.5523099999999999
askhigh=1.55915
asklow=1.5436299999999998
last=0.0
lasthigh=0.0
lastlow=0.0
volume_real=0.0
volumehigh_real=0.0
volumelow_real=0.0
option_strike=0.0
point=1e-05
trade_tick_value=0.7043642408362214
trade_tick_value_profit=0.7043642408362214
trade_tick_value_loss=0.7044535553770941
trade_tick_size=1e-05
trade_contract_size=100000.0
trade_accrued_interest=0.0
trade_face_value=0.0
trade_liquidity_rate=0.0
volume_min=0.01
volume_max=500.0
volume_step=0.01
volume_limit=0.0
swap_long=-1.1
swap_short=-0.9
margin_initial=0.0
margin_maintenance=0.0
session_volume=0.0
session_turnover=0.0
session_interest=0.0
session_buy_orders_volume=0.0
session_sell_orders_volume=0.0
session_open=0.0
session_close=0.0
session_aw=0.0
session_price_settlement=0.0
session_price_limit_min=0.0
session_price_limit_max=0.0
margin_hedged=100000.0
price_change=0.0
price_volatility=0.0
price_theoretical=0.0
price_greeks_delta=0.0
price_greeks_theta=0.0
price_greeks_gamma=0.0
```

© 2000-2025, MetaQuotes Ltd.

```
  price_greeks_vega=0.0
  price_greeks_rho=0.0
  price_greeks_omega=0.0
  price_sensitivity=0.0
  basis=
  category=
  currency_base=EUR
  currency_profit=CAD
  currency_margin=EUR
  bank=
  description=Euro vs Canadian Dollar
  exchange=
  formula=
  isin=
  name=EURCAD
  page=http://www.google.com/finance?q=EURCAD
  path=Forex\EURCAD

symbol_info_dict() as dataframe:
         property                                      value
0          custom                                      False
1      chart_mode                                          0
2          select                                       True
3         visible                                       True
4   session_deals                                          0
..            ...                                       ...
91        formula
92           isin
93           name                                     EURCAD
94           page  http://www.google.com/finance?q=EURCAD
95           path                                Forex\EURCAD

[96 rows x 2 columns]
```

另见

[symbol_info](#)

© 2000-2025, MetaQuotes Ltd.

# market_book_add

订阅MetaTrader5程序端的指定交易品种的市场深度更改事件。

```
market_book_add(
   symbol      // 交易品种名称
)
```

*交易品种*

[in] 交易品种名称。所需的未命名参数。

返回值

如果成功返回true，否则 false。

注意

该函数类似于MarketBookAdd。

另见

market_book_get, market_book_release, 市场深度结构

© 2000-2025, MetaQuotes Ltd.

# market_book_get

从BookInfo返回一个元组，其中包含指定交易品种的市场深度条目。

```
market_book_get(
   symbol      // 交易品种名称
)
```

*交易品种*

[in] 交易品种名称。所需的未命名参数。

返回值

从BookInfo条目中以元组形式返回市场深度内容，其中包含订单类型、价格和手数交易量。BookInfo类似于MqlBookInfo结构。

错误情况下返回None。可使用last_error()获取错误信息。

注意

应使用market_book_add()函数预先执行订阅市场深度更改事件。

该函数类似于MarketBookGet。

例如：

```
import MetaTrader5 as mt5
import time
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print("")

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
   # 断开与MetaTrader 5程序端的连接
    mt5.shutdown()
    quit()

# 订阅EURUSD (市场深度)的市场深度更新
if mt5.market_book_add('EURUSD'):
  # 循环获取10次市场深度数据
   for i in range(10):
        # 获取市场深度内容 (市场深度)
        items = mt5.market_book_get('EURUSD')
        # 以单个字符串'as is' 保持原来状态)显示整个市场深度
        print(items)
        # 现在为了更加清晰而分别显示每个订单
        if items:
            for it in items:
```

© 2000-2025, MetaQuotes Ltd.

```
                # order content
                print(it._asdict())
        # 暂停5秒后再请求下一个市场深度数据
        time.sleep(5)
  # 取消订阅市场深度更新（市场深度）
   mt5.market_book_release('EURUSD')
else:
    print("mt5.market_book_add('EURUSD') failed, error code =",mt5.last_error())

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.34

(BookInfo(type=1, price=1.20038, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20038, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20032, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.2003, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20028, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20026, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20025, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20023, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20017, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.2004299999999999, volume=250, volume_dbl=250.0), BookInfo(ty
{'type': 1, 'price': 1.2004299999999999, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20037, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20036, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20034, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20031, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20029, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20028, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20022, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.2004299999999999, volume=250, volume_dbl=250.0), BookInfo(ty
{'type': 1, 'price': 1.2004299999999999, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20037, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20036, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20034, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20031, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20029, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20028, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20022, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20036, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20036, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20029, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20028, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20026, 'volume': 36, 'volume_dbl': 36.0}
```

© 2000-2025, MetaQuotes Ltd.

```
{'type': 2, 'price': 1.20023, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20022, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20021, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20014, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20035, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20035, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20029, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20027, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20025, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20023, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20022, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20021, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20014, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20037, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20037, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20031, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.2003, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20028, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20025, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20023, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20022, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20016, 'volume': 250, 'volume_dbl': 250.0}
```

另见

market_book_add, market_book_release, 市场深度结构

© 2000-2025, MetaQuotes Ltd.

# market_book_release

取消订阅MetaTrader5程序端的指定交易品种的市场深度更改事件。

```
market_book_release(
   symbol      // 交易品种名称
)
```

*交易品种*

[in] 交易品种名称。所需的未命名参数。

返回值

如果成功返回true，否则 false。

注意

该函数类似于MarketBookRelease。

另见

market_book_add, market_book_get, 市场深度结构

© 2000-2025, MetaQuotes Ltd.

# copy_rates_from

从指定日期开始从MetaTrader 5程序端获取柱形图。

```
copy_rates_from(
   symbol,       // 交易品种名称
   timeframe,    // 时间周期
   date_from,    // 初始柱形图开盘日期
   count         // 柱形图数
   )
```

参数

*symbol*

[in] 交易品种名称，例如"EURUSD"。所需的未命名参数。

*timeframe*

[in] 请求柱形图的时间周期。根据TIMEFRAME枚举的值设置。所需的未命名参数。

*date_from*

[in] 请求样本中的第一个柱形图的开盘日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*count*

[in] 将要接收的柱形图数。所需的未命名参数。

返回值

返回numpy数据柱形图（包含以下指定列：时间、开盘价、最高价、最低价、收盘价、tick_volume、点差和real_volume）。错误情况下返回None。可使用last_error()获取错误信息。

注意

参见CopyRates()函数获取更多信息。

只有日期小于（早于）或等于指定日期的数据。这意味着，任何柱的开盘时间始终少于或等于指定的值。

MetaTrader 5程序端仅在用户可用的图表历史记录中提供柱形图。可供用户使用的柱形图数量设置在"图表中最大数量(Max.)柱形图"参数中。

当创建'datetime'对象时，Python使用本地时区，而MetaTrader 5以UTC时区保存报价和柱形图开盘时间（没有时移）。因此，'datetime'应在UTC时间内创建，用于执行使用时间的函数。从MetaTrader 5程序端接收到的数据使用UTC时间。

**TIMEFRAME** 是一个包含可能图表周期值的枚举

| ID | 描述 |
|---|---|
| TIMEFRAME_M1 | 1分钟 |
| TIMEFRAME_M2 | 2分钟 |
| TIMEFRAME_M3 | 3分钟 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| TIMEFRAME_M4 | 4 分钟 |
| TIMEFRAME_M5 | 5 分钟 |
| TIMEFRAME_M6 | 6 分钟 |
| TIMEFRAME_M10 | 10 分钟 |
| TIMEFRAME_M12 | 12 分钟 |
| TIMEFRAME_M12 | 15 分钟 |
| TIMEFRAME_M20 | 20 分钟 |
| TIMEFRAME_M30 | 30 分钟 |
| TIMEFRAME_H1 | 1 小时 |
| TIMEFRAME_H2 | 2 小时 |
| TIMEFRAME_H3 | 3 小时 |
| TIMEFRAME_H4 | 4 小时 |
| TIMEFRAME_H6 | 6 小时 |
| TIMEFRAME_H8 | 8 小时 |
| TIMEFRAME_H12 | 12 小时 |
| TIMEFRAME_D1 | 1 天 |
| TIMEFRAME_W1 | 1 周 |
| TIMEFRAME_MN1 | 1 个月 |

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 导入'pandas'模块,用于以表格形式显示获得的数据
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 导入用于处理时区的pytz模块
import pytz

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
```

© 2000-2025, MetaQuotes Ltd.

```
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 将时区设置为UTC
timezone = pytz.timezone("Etc/UTC")
# 以UTC时区创建'datetime'对象，以避免实现本地时区偏移
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
# 在UTC时区，获取01.10.2020开始的10个EURUSD H4柱形图
rates = mt5.copy_rates_from("EURUSD", mt5.TIMEFRAME_H4, utc_from, 10)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
# 在新行显示所获得数据的每个元素
print("Display obtained data 'as is'")
for rate in rates:
    print(rate)

# 从所获得的数据创建DataFrame
rates_frame = pd.DataFrame(rates)
# 将时间（以秒为单位）转换为日期时间格式
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# 显示数据
print("\nDisplay dataframe with data")
print(rates_frame)
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

将获得的数据显示为“保持原来状态”
(1578484800, 1.11382, 1.11385, 1.1111, 1.11199, 9354, 1, 0)
(1578499200, 1.11199, 1.11308, 1.11086, 1.11179, 10641, 1, 0)
(1578513600, 1.11178, 1.11178, 1.11016, 1.11053, 4806, 1, 0)
(1578528000, 1.11053, 1.11193, 1.11033, 1.11173, 3480, 1, 0)
(1578542400, 1.11173, 1.11189, 1.11126, 1.11182, 2236, 1, 0)
(1578556800, 1.11181, 1.11203, 1.10983, 1.10993, 7984, 1, 0)
(1578571200, 1.10994, 1.11173, 1.10965, 1.11148, 7406, 1, 0)
(1578585600, 1.11149, 1.11149, 1.10923, 1.11046, 7468, 1, 0)
(1578600000, 1.11046, 1.11097, 1.11033, 1.11051, 3450, 1, 0)
(1578614400, 1.11051, 1.11093, 1.11017, 1.11041, 2448, 1, 0)

显示带有数据的数据框
                 time     open     high      low    close  tick_volume  spread  real_v
0 2020-01-08 12:00:00  1.11382  1.11385  1.11110  1.11199         9354       1
1 2020-01-08 16:00:00  1.11199  1.11308  1.11086  1.11179        10641       1
2 2020-01-08 20:00:00  1.11178  1.11178  1.11016  1.11053         4806       1
3 2020-01-09 00:00:00  1.11053  1.11193  1.11033  1.11173         3480       1
```

© 2000-2025, MetaQuotes Ltd.

```
4 2020-01-09 04:00:00  1.11173  1.11189  1.11126  1.11182         2236       1
5 2020-01-09 08:00:00  1.11181  1.11203  1.10983  1.10993         7984       1
6 2020-01-09 12:00:00  1.10994  1.11173  1.10965  1.11148         7406       1
7 2020-01-09 16:00:00  1.11149  1.11149  1.10923  1.11046         7468       1
8 2020-01-09 20:00:00  1.11046  1.11097  1.11033  1.11051         3450       1
9 2020-01-10 00:00:00  1.11051  1.11093  1.11017  1.11041         2448       1
```

另见

CopyRates、copy_rates_from_pos、copy_rates_range、copy_ticks_from、copy_ticks_range

© 2000-2025, MetaQuotes Ltd.

# copy_rates_from_pos

从指定索引开始从MetaTrader 5程序端获取柱形图。

```
copy_rates_from_pos(
   symbol,       // 交易品种名称
   timeframe,    // 时间周期
   start_pos,    //初始柱形图索引
   count         // 柱形图数
   )
```

参数

*symbol*

[in] 交易品种名称，例如"EURUSD"。所需的未命名参数。

*timeframe*

[in] 请求柱形图的时间周期。根据TIMEFRAME枚举的值设置。所需的未命名参数。

*start_pos*

[in] 请求数据的初始柱形图索引。柱形图变化从现在到过去。因此，零柱形图意味着当前柱形图。所需的未命名参数。

*count*

[in] 将要接收的柱形图数。所需的未命名参数。

返回值

返回numpy数据柱形图（包含以下指定列：时间、开盘价、最高价、最低价、收盘价、tick_volume、点差和real_volume）。错误情况下返回None。可使用last_error()获取错误信息。

注意

参见CopyRates()函数获取更多信息。

MetaTrader 5程序端仅在用户可用的图表历史记录中提供柱形图。可供用户使用的柱形图数量设置在“图表中最大数量(Max.)柱形图”参数中。

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 导入'pandas'模块，用于以表格形式显示获得的数据
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
```

© 2000-2025, MetaQuotes Ltd.

```
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 从当日获取10个GBPUSD D1柱形图
rates = mt5.copy_rates_from_pos("GBPUSD", mt5.TIMEFRAME_D1, 0, 10)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
# 在新行显示所获得数据的每个元素
print("Display obtained data 'as is'")
for rate in rates:
    print(rate)

# 从所获得的数据创建DataFrame
rates_frame = pd.DataFrame(rates)
# 将时间（以秒为单位）转换为日期时间格式
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# 显示数据
print("\nDisplay dataframe with data")
print(rates_frame)
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

将获得的数据显示为“保持原来状态”
(1581552000, 1.29568, 1.30692, 1.29441, 1.30412, 68228, 0, 0)
(1581638400, 1.30385, 1.30631, 1.3001, 1.30471, 56498, 0, 0)
(1581897600, 1.30324, 1.30536, 1.29975, 1.30039, 49400, 0, 0)
(1581984000, 1.30039, 1.30486, 1.29705, 1.29952, 62288, 0, 0)
(1582070400, 1.29952, 1.3023, 1.29075, 1.29187, 57909, 0, 0)
(1582156800, 1.29186, 1.29281, 1.28489, 1.28792, 61033, 0, 0)
(1582243200, 1.28802, 1.29805, 1.28746, 1.29566, 66386, 0, 0)
(1582502400, 1.29426, 1.29547, 1.28865, 1.29283, 66933, 0, 0)
(1582588800, 1.2929, 1.30178, 1.29142, 1.30037, 80121, 0, 0)
(1582675200, 1.30036, 1.30078, 1.29136, 1.29374, 49286, 0, 0)

显示带有数据的数据框
        time     open     high      low    close  tick_volume  spread  real_volume
0 2020-02-13  1.29568  1.30692  1.29441  1.30412        68228       0            0
1 2020-02-14  1.30385  1.30631  1.30010  1.30471        56498       0            0
2 2020-02-17  1.30324  1.30536  1.29975  1.30039        49400       0            0
3 2020-02-18  1.30039  1.30486  1.29705  1.29952        62288       0            0
4 2020-02-19  1.29952  1.30230  1.29075  1.29187        57909       0            0
5 2020-02-20  1.29186  1.29281  1.28489  1.28792        61033       0            0
6 2020-02-21  1.28802  1.29805  1.28746  1.29566        66386       0            0
7 2020-02-24  1.29426  1.29547  1.28865  1.29283        66933       0            0
8 2020-02-25  1.29290  1.30178  1.29142  1.30037        80121       0            0
```

© 2000-2025, MetaQuotes Ltd.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 2020-02-26 | 1.30036 | 1.30078 | 1.29136 | 1.29374 | 49286 | 0 | 0 |

另见

CopyRates、copy_rates_from、copy_rates_range、copy_ticks_from、copy_ticks_range

© 2000-2025, MetaQuotes Ltd.

# copy_rates_range

从MetaTrader 5程序端获取指定日期范围内的柱形图。

```
copy_rates_range(
   symbol,       // 交易品种名称
   timeframe,    // 时间周期
   date_from,    // 请求柱形图的开始日期
   date_to       // 请求柱形图的结束日期
   )
```

参数

*symbol*

[in] 交易品种名称，例如"EURUSD"。所需的未命名参数。

*timeframe*

[in] 请求柱形图的时间周期。根据TIMEFRAME枚举的值设置。所需的未命名参数。

*date_from*

[in] 请求柱形图的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。包括开盘时间的柱形图 >= 返回date_from。所需的未命名参数。

*date_to*

[in] 请求柱形图的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。包括开盘时间的柱形图 <= 返回date_to。所需的未命名参数。

返回值

返回numpy数据柱形图（包含以下指定列：时间、开盘价、最高价、最低价、收盘价、tick_volume、点差和real_volume）。错误情况下返回None。可使用last_error()获取错误信息。

注意

参见CopyRates()函数获取更多信息。

MetaTrader 5程序端仅在用户可用的图表历史记录中提供柱形图。可供用户使用的柱形图数量设置在"图表中最大数量(Max.)柱形图"参数中。

当创建'datetime'对象时，Python使用本地时区，而MetaTrader 5以UTC时区保存报价和柱形图开盘时间（没有时移）。 因此，'datetime'应在UTC时间内创建，用于执行使用时间的函数。从MetaTrader 5程序端接收到的数据使用UTC时间。

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
```

© 2000-2025, MetaQuotes Ltd.

```
# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' objects in UTC time zone to avoid the implementation of a local ti
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
utc_to = datetime(2020, 1, 11, hour = 13, tzinfo=timezone)
# get bars from USDJPY M5 within the interval of 2020.01.10 00:00 - 2020.01.11 13:00 i
rates = mt5.copy_rates_range("USDJPY", mt5.TIMEFRAME_M5, utc_from, utc_to)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display each element of obtained data in a new line
print("Display obtained data 'as is'")
counter=0
for rate in rates:
    counter+=1
    if counter<=10:
        print(rate)

# create DataFrame out of the obtained data
rates_frame = pd.DataFrame(rates)
# convert time in seconds into the 'datetime' format
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with data")
print(rates_frame.head(10))
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

将获得的数据显示为“保持原来状态”
(1578614400, 109.513, 109.527, 109.505, 109.521, 43, 2, 0)
(1578614700, 109.521, 109.549, 109.518, 109.543, 215, 8, 0)
(1578615000, 109.543, 109.543, 109.466, 109.505, 98, 10, 0)
(1578615300, 109.504, 109.534, 109.502, 109.517, 155, 8, 0)
(1578615600, 109.517, 109.539, 109.513, 109.527, 71, 4, 0)
(1578615900, 109.526, 109.537, 109.484, 109.52, 106, 9, 0)
(1578616200, 109.52, 109.524, 109.508, 109.51, 205, 7, 0)
(1578616500, 109.51, 109.51, 109.491, 109.496, 44, 8, 0)
(1578616800, 109.496, 109.509, 109.487, 109.5, 85, 5, 0)
(1578617100, 109.5, 109.504, 109.487, 109.489, 82, 7, 0)
```

© 2000-2025, MetaQuotes Ltd.

显示带有数据的数据框

| | time | open | high | low | close | tick_volume | spread | real_v |
|---|---|---|---|---|---|---|---|---|
| 0 | 2020-01-10 00:00:00 | 109.513 | 109.527 | 109.505 | 109.521 | 43 | 2 | |
| 1 | 2020-01-10 00:05:00 | 109.521 | 109.549 | 109.518 | 109.543 | 215 | 8 | |
| 2 | 2020-01-10 00:10:00 | 109.543 | 109.543 | 109.466 | 109.505 | 98 | 10 | |
| 3 | 2020-01-10 00:15:00 | 109.504 | 109.534 | 109.502 | 109.517 | 155 | 8 | |
| 4 | 2020-01-10 00:20:00 | 109.517 | 109.539 | 109.513 | 109.527 | 71 | 4 | |
| 5 | 2020-01-10 00:25:00 | 109.526 | 109.537 | 109.484 | 109.520 | 106 | 9 | |
| 6 | 2020-01-10 00:30:00 | 109.520 | 109.524 | 109.508 | 109.510 | 205 | 7 | |
| 7 | 2020-01-10 00:35:00 | 109.510 | 109.510 | 109.491 | 109.496 | 44 | 8 | |
| 8 | 2020-01-10 00:40:00 | 109.496 | 109.509 | 109.487 | 109.500 | 85 | 5 | |
| 9 | 2020-01-10 00:45:00 | 109.500 | 109.504 | 109.487 | 109.489 | 82 | 7 | |

另 见

CopyRates、copy_rates_from、copy_rates_range、copy_ticks_from、copy_ticks_range

© 2000-2025, MetaQuotes Ltd.

# copy_ticks_from

从指定日期开始从MetaTrader 5程序端获取报价。

```
copy_ticks_from(
   symbol,       // 交易品种名称
   date_from,    // 请求报价的日期
   count,        // 请求的报价数
   flags         // 定义请求报价类型的标识组合
   )
```

参数

*symbol*

[in] 交易品种名称，例如"EURUSD"。所需的未命名参数。

*date_from*

[in] 请求报价的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*count*

[in] 将要接收的报价数。所需的未命名参数。

*flags*

[in] 确定请求报价类型的标记。COPY_TICKS_INFO - 卖价和/或买价变化的报价，COPY_TICKS_TRADE -最后价和交易量变化的报价，COPY_TICKS_ALL -全部报价。标识值在COPY_TICKS枚举中描述。所需的未命名参数。

返回值

返回numpy数据报价（包含以下指定列：时间、卖价、买价、最后价、标识）。'flags'值可以是TICK_FLAG枚举中的标识组合。错误情况下返回None。可使用last_error()获取错误信息。

注意

参见CopyTicks函数获取更多信息。

当创建'datetime'对象时，Python使用本地时区，而MetaTrader 5以UTC时区保存报价和柱形图开盘时间（没有时移）。因此，'datetime'应在UTC时间内创建，用于执行使用时间的函数。从MetaTrader 5程序端接收到的数据使用UTC时间。

COPY_TICKS定义了可以使用copy_ticks_from()和copy_ticks_range()函数请求的报价类型。

| ID | 描述 |
| --- | --- |
| COPY_TICKS_ALL | 所有报价 |
| COPY_TICKS_INFO | 包含卖价（Bid）和/或买价（Ask）价格变化的报价 |
| COPY_TICKS_TRADE | 包含最后价（Last）和/或交易量（Volume）价格变化的报价 |

© 2000-2025, MetaQuotes Ltd.

TICK_FLAG 定义可能的报价标识。这些标识用于描述通过copy_ticks_from()和copy_ticks_range()函数获得的报价。

| ID | 描述 |
| --- | --- |
| TICK_FLAG_BID | 卖价 (Bid)已更改 |
| TICK_FLAG_ASK | 买价 (Ask)已更改 |
| TICK_FLAG_LAST | 最后价 (Last)已更改 |
| TICK_FLAG_VOLUME | 交易量已更改 |
| TICK_FLAG_BUY | 最近 "买入"价格已更改 |
| TICK_FLAG_SELL | 最近 "卖出"价格已更改 |

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 导入'pandas'模块,用于以表格形式显示获得的数据
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 导入用于处理时区的pytz模块
import pytz

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' object in UTC time zone to avoid the implementation of a local tir
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
# request 100 000 EURUSD ticks starting from 10.01.2019 in UTC time zone
ticks = mt5.copy_ticks_from("EURUSD", utc_from, 100000, mt5.COPY_TICKS_ALL)
print("Ticks received:",len(ticks))

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display data on each tick on a new line
print("Display obtained ticks 'as is'")
count = 0
for tick in ticks:
```

© 2000-2025, MetaQuotes Ltd.

```
    count+=1
    print(tick)
    if count >= 10:
        break

# create DataFrame out of the obtained data
ticks_frame = pd.DataFrame(ticks)
# 将时间（以秒为单位）转换为日期时间格式
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with ticks")
print(ticks_frame.head(10))
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

已接收报价:100000
将获得的报价显示为“保持原来状态”
(1578614400, 1.11051, 1.11069, 0., 0, 1578614400987, 134, 0.)
(1578614402, 1.11049, 1.11067, 0., 0, 1578614402025, 134, 0.)
(1578614404, 1.1105, 1.11066, 0., 0, 1578614404057, 134, 0.)
(1578614404, 1.11049, 1.11067, 0., 0, 1578614404344, 134, 0.)
(1578614412, 1.11052, 1.11064, 0., 0, 1578614412106, 134, 0.)
(1578614418, 1.11039, 1.11051, 0., 0, 1578614418265, 134, 0.)
(1578614418, 1.1104, 1.1105, 0., 0, 1578614418905, 134, 0.)
(1578614419, 1.11039, 1.11051, 0., 0, 1578614419519, 134, 0.)
(1578614456, 1.11037, 1.11065, 0., 0, 1578614456011, 134, 0.)
(1578614456, 1.11039, 1.11051, 0., 0, 1578614456015, 134, 0.)

显示带有报价的数据框
                 time      bid      ask  last  volume       time_msc  flags  volume_re
0 2020-01-10 00:00:00  1.11051  1.11069   0.0       0  1578614400987    134          0
1 2020-01-10 00:00:02  1.11049  1.11067   0.0       0  1578614402025    134          0
2 2020-01-10 00:00:04  1.11050  1.11066   0.0       0  1578614404057    134          0
3 2020-01-10 00:00:04  1.11049  1.11067   0.0       0  1578614404344    134          0
4 2020-01-10 00:00:12  1.11052  1.11064   0.0       0  1578614412106    134          0
5 2020-01-10 00:00:18  1.11039  1.11051   0.0       0  1578614418265    134          0
6 2020-01-10 00:00:18  1.11040  1.11050   0.0       0  1578614418905    134          0
7 2020-01-10 00:00:19  1.11039  1.11051   0.0       0  1578614419519    134          0
8 2020-01-10 00:00:56  1.11037  1.11065   0.0       0  1578614456011    134          0
9 2020-01-10 00:00:56  1.11039  1.11051   0.0       0  1578614456015    134          0
```

另见

CopyRates、copy_rates_from_pos、copy_rates_range、copy_ticks_from、copy_ticks_range

© 2000-2025, MetaQuotes Ltd.

# copy_ticks_range

从MetaTrader 5程序端获取指定日期范围内的报价。

```
copy_ticks_range(
   symbol,       // 交易品种名称
   date_from,    // 请求报价的日期
   date_to,      // 请求报价的结束日期
   flags         // 定义请求报价类型的标识组合
   )
```

参数

*symbol*

[in] 交易品种名称，例如"EURUSD"。所需的未命名参数。

*date_from*

[in] 请求报价的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*date_to*

[in] 请求报价的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*flags*

[in] 确定请求报价类型的标记。COPY_TICKS_INFO - 卖价和/或买价变化的报价，COPY_TICKS_TRADE -最后价和交易量变化的报价，COPY_TICKS_ALL -全部报价。标识值在COPY_TICKS枚举中描述。所需的未命名参数。

返回值

返回numpy数据报价（包含以下指定列：时间、卖价、买价、最后价、标识）。'flags'值可以是TICK_FLAG枚举中的标识组合。错误情况下返回None。可使用last_error()获取错误信息。

注意

参见CopyTicks函数获取更多信息。

当创建'datetime'对象时，Python使用本地时区，而MetaTrader 5以UTC时区保存报价和柱形图开盘时间（没有时移）。 因此，'datetime'应在UTC时间内创建，用于执行使用时间的函数。从MetaTrader 5获得的数据有UTC时间，但Python在尝试打印时再次应用本地时移。 因此，所获得的数据也应进行校正，以便直观表示。

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
```

© 2000-2025, MetaQuotes Ltd.

```
# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' objects in UTC time zone to avoid the implementation of a local ti
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
utc_to = datetime(2020, 1, 11, tzinfo=timezone)
# request AUDUSD ticks within 11.01.2020 - 11.01.2020
ticks = mt5.copy_ticks_range("AUDUSD", utc_from, utc_to, mt5.COPY_TICKS_ALL)
print("Ticks received:",len(ticks))

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display data on each tick on a new line
print("Display obtained ticks 'as is'")
count = 0
for tick in ticks:
    count+=1
    print(tick)
    if count >= 10:
        break

# create DataFrame out of the obtained data
ticks_frame = pd.DataFrame(ticks)
# convert time in seconds into the datetime format
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with ticks")
print(ticks_frame.head(10))
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

已接收报价:37008
将获得的报价显示为“保持原来状态”
(1578614400, 0.68577, 0.68594, 0., 0, 1578614400820, 134, 0.)
(1578614401, 0.68578, 0.68594, 0., 0, 1578614401128, 130, 0.)
(1578614401, 0.68575, 0.68594, 0., 0, 1578614401128, 130, 0.)
(1578614411, 0.68576, 0.68594, 0., 0, 1578614411388, 130, 0.)
(1578614411, 0.68575, 0.68594, 0., 0, 1578614411560, 130, 0.)
(1578614414, 0.68576, 0.68595, 0., 0, 1578614414973, 134, 0.)
(1578614430, 0.68576, 0.68594, 0., 0, 1578614430188, 4, 0.)
(1578614450, 0.68576, 0.68595, 0., 0, 1578614450408, 4, 0.)
```

© 2000-2025, MetaQuotes Ltd.

```
(1578614450, 0.68576, 0.68594, 0., 0, 1578614450519, 4, 0.)
(1578614456, 0.68575, 0.68594, 0., 0, 1578614456363, 130, 0.)

显示带有报价的数据框
                 time      bid      ask  last  volume       time_msc  flags  volume_re
0 2020-01-10 00:00:00  0.68577  0.68594   0.0       0  1578614400820    134          0
1 2020-01-10 00:00:01  0.68578  0.68594   0.0       0  1578614401128    130          0
2 2020-01-10 00:00:01  0.68575  0.68594   0.0       0  1578614401128    130          0
3 2020-01-10 00:00:11  0.68576  0.68594   0.0       0  1578614411388    130          0
4 2020-01-10 00:00:11  0.68575  0.68594   0.0       0  1578614411560    130          0
5 2020-01-10 00:00:14  0.68576  0.68595   0.0       0  1578614414973    134          0
6 2020-01-10 00:00:30  0.68576  0.68594   0.0       0  1578614430188      4          0
7 2020-01-10 00:00:50  0.68576  0.68595   0.0       0  1578614450408      4          0
8 2020-01-10 00:00:50  0.68576  0.68594   0.0       0  1578614450519      4          0
9 2020-01-10 00:00:56  0.68575  0.68594   0.0       0  1578614456363    130          0
```

另见

CopyRates、copy_rates_from_pos、copy_rates_range、copy_ticks_from、copy_ticks_range

© 2000-2025, MetaQuotes Ltd.

# terminal_info

获取已连接的MetaTrader 5客户端的状态和设置。

```
terminal_info()
```

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数返回在一次调用中使用TerminalInfoInteger、 TerminalInfoDouble和TerminalInfoDouble可获得的所有数据。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示有关MetaTrader 5版本的数据
print(mt5.version())
# 显示有关程序端设置和状态的信息
terminal_info=mt5.terminal_info()
if terminal_info!=None:
    # display the terminal data 'as is'
    print(terminal_info)
    # display data in the form of a list
    print("Show terminal_info()._asdict():")
    terminal_info_dict = mt5.terminal_info()._asdict()
    for prop in terminal_info_dict:
        print("  {}={}".format(prop, terminal_info_dict[prop]))
    print()
   # 将词典转换为DataFrame和print
    df=pd.DataFrame(list(terminal_info_dict.items()),columns=['property','value'])
    print("terminal_info() as dataframe:")
    print(df)

# 断开与MetaTrader 5程序端的连接
```

© 2000-2025, MetaQuotes Ltd.

```
mt5.shutdown()


结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
[500, 2366, '20 Mar 2020']
TerminalInfo(community_account=True, community_connection=True, connected=True,....
Show terminal_info()._asdict():
  community_account=True
  community_connection=True
  connected=True
  dlls_allowed=False
  trade_allowed=False
  tradeapi_disabled=False
  email_enabled=False
  ftp_enabled=False
  notifications_enabled=False
  mqid=False
  build=2366
  maxbars=5000
  codepage=1251
  ping_last=77850
  community_balance=707.10668201585
  retransmission=0.0
  company=MetaQuotes Software Corp.
  name=MetaTrader 5
  language=Russian
  path=E:\ProgramFiles\MetaTrader 5
  data_path=E:\ProgramFiles\MetaTrader 5
  commondata_path=C:\Users\Rosh\AppData\Roaming\MetaQuotes\Terminal\Common

terminal_info() as dataframe:
                  property                      value
0       community_account                       True
1    community_connection                       True
2               connected                       True
3            dlls_allowed                      False
4           trade_allowed                      False
5       tradeapi_disabled                      False
6           email_enabled                      False
7             ftp_enabled                      False
8   notifications_enabled                      False
9                    mqid                      False
10                  build                       2367
11                maxbars                       5000
12               codepage                       1251
13             ping_last                      80953
14      community_balance                    707.107
```

© 2000-2025, MetaQuotes Ltd.

```
15         retransmission                   0.063593
16                company  MetaQuotes Software Corp.
17                   name                MetaTrader 5
18               language                     Russian
```

另见

initialize、shutdown、version

© 2000-2025, MetaQuotes Ltd.

# orders_total

获取活动订单的数量。

```
orders_total()
```

返回值

整数值。

注意

该函数类似于OrdersTotal。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 检查是否存在活动订单
orders=mt5.orders_total()
if orders>0:
    print("Total orders=",orders)
其他:
    print("Orders not found")

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另见

orders_get、positions_total

© 2000-2025, MetaQuotes Ltd.

# orders_get

获取可通过交易品种或单号筛选的活动订单。共有三个调用选项。

无参数调用。返回所有交易品种的活动订单。

```
orders_get()
```

调用指定应接收的交易品种活动订单。

```
orders_get(
   symbol="SYMBOL"       //交易品种名称
)
```

调用指定应接收的交易品种组活动订单。

```
orders_get(
   group="GROUP"         // 用于选择交易品种订单的过滤器
)
```

调用指定订单单号。

```
orders_get(
   ticket=TICKET         // 单号
)
```

*symbol="SYMBOL"*

[in] 交易品种名称。可选的命名参数。如果指定交易品种，则忽略*ticket*参数。

*group="GROUP"*

[in] 用于跑了一组必要交易品种的过滤器。可选的命名参数。如果指定了组，则函数只返回满足交易品种名称指定条件的活动订单。

*ticket=TICKET*

[in] 订单单号 (ORDER_TICKET)。可选的命名参数。

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数允许在一个类似于OrdersTotal和OrderSelect的串联调用中接收所有活动订单。

*group*参数可以根据交易品种对订单进行分类。'*'可以用在字符串的开头和结尾。

*group*参数可以包含多个逗号分隔条件。条件可以使用'*'设置为掩码。逻辑否定字符'!'可以用来表示排除条件。所有条件依序应用，这意味着应该先指定包含到组中的条件，然后再指定排除条件。例如，*group*="*, !EUR"意味着应该先选择所有交易品种的订单，然后排除交易品种名称中包含"EUR"的订单。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
```

© 2000-2025, MetaQuotes Ltd.

```
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 显示GBPUSD活动订单的数据
orders=mt5.orders_get(symbol="GBPUSD")
if orders is None:
    print("No orders on GBPUSD, error code={}".format(mt5.last_error()))
else:
    print("Total orders on GBPUSD:",len(orders))
    # display all active orders
    for order in orders:
        print(order)
print()

# 获取名称包含"*GBP*"的交易品种的订单列表
gbp_orders=mt5.orders_get(group="*GBP*")
if gbp_orders is None:
    print("No orders with group=\"*GBP*\", error code={}".format(mt5.last_error()))
else:
    print("orders_get(group=\"*GBP*\")={}".format(len(gbp_orders)))
    # display these orders as a table using pandas.DataFrame
    df=pd.DataFrame(list(gbp_orders),columns=gbp_orders[0]._asdict().keys())
    df.drop(['time_done', 'time_done_msc', 'position_id', 'position_by_id', 'reason',
    df['time_setup'] = pd.to_datetime(df['time_setup'], unit='s')
    print(df)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

Total orders on GBPUSD: 2
TradeOrder(ticket=554733548, time_setup=1585153667, time_setup_msc=1585153667718, time
TradeOrder(ticket=554733621, time_setup=1585153671, time_setup_msc=1585153671419, time

orders_get(group="*GBP*")=4
      ticket           time_setup  time_setup_msc  time_expiration  type  type_time  ty
0  554733548 2020-03-25 16:27:47   1585153667718                0     3          0
1  554733621 2020-03-25 16:27:51   1585153671419                0     2          0
```

© 2000-2025, MetaQuotes Ltd.

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 554746664 | 2020-03-25 16:38:14 | 1585154294401 | 0 | 3 | 0 |
| 3 | 554746710 | 2020-03-25 16:38:17 | 1585154297022 | 0 | 2 | 0 |

另见

orders_total、positions_get

© 2000-2025, MetaQuotes Ltd.

# order_calc_margin

返回预付款（用账户货币表示）来执行指定的交易操作。

```
order_calc_margin(
   action,      // 订单类型（ORDER_TYPE_BUY or ORDER_TYPE_SELL）
   symbol,      // 交易品种名称
   volume,      // 交易量
   price        // 开盘价
   )
```

参数

*action*

[in] 从ORDER_TYPE枚举中获取值的订单类型。所需的未命名参数。

*symbol*

[in] 交易品种名称。所需的未命名参数。

*volume*

[in] 交易操作的交易量。所需的未命名参数。

*price*

[in] 开盘价。所需的未命名参数。

返回值

如果成功返回真实值，否则返回None。可以使用last_error()获取错误信息。

注意

该功能允许评估当前账户和当前市场环境中指定订单类型所需的预付款，而无需考虑当前挂单和未结持仓。该函数类似于OrderCalcMargin。

ORDER_TYPE

| ID | 描述 |
|---|---|
| ORDER_TYPE_BUY | 市价买单 |
| ORDER_TYPE_SELL | 市价卖单 |
| ORDER_TYPE_BUY_LIMIT | Buy Limit挂单 |
| ORDER_TYPE_SELL_LIMIT | SellLimit挂单 |
| ORDER_TYPE_BUY_STOP | Buy Stop挂单 |
| ORDER_TYPE_SELL_STOP | SellStop挂单 |
| ORDER_TYPE_BUY_STOP_LIMIT | 在达到订单价格时，以StopLimit价格下单Buy Limit挂单 |
| ORDER_TYPE_SELL_STOP_LIMIT | 在达到订单价格时，以StopLimit价格下单SellLimit挂单 |
| ORDER_TYPE_CLOSE_BY | 用于反向平仓的订单 |

例如：

© 2000-2025, MetaQuotes Ltd.

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# arrange the symbol list
symbols=("EURUSD","GBPUSD","USDJPY", "USDCHF","EURJPY","GBPJPY")
print("Symbols to check margin:", symbols)
action=mt5.ORDER_TYPE_BUY
lot=0.1
for symbol in symbols:
    symbol_info=mt5.symbol_info(symbol)
    if symbol_info is None:
        print(symbol,"not found, skipped")
        continue
    if not symbol_info.visible:
        print(symbol, "is not visible, trying to switch on")
        if not mt5.symbol_select(symbol,True):
            print("symbol_select({}}) failed, skipped",symbol)
            continue
    ask=mt5.symbol_info_tick(symbol).ask
    margin=mt5.order_calc_margin(action,symbol,lot,ask)
    if margin != None:
        print("   {} buy {} lot margin: {} {}".format(symbol,lot,margin,account_currer
    else:
        print("order_calc_margin failed: , error code =", mt5.last_error())

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currency: USD

Symbols to check margin: ('EURUSD', 'GBPUSD', 'USDJPY', 'USDCHF', 'EURJPY', 'GBPJPY')
   EURUSD buy 0.1 lot margin: 109.91 USD
   GBPUSD buy 0.1 lot margin: 122.73 USD
   USDJPY buy 0.1 lot margin: 100.0 USD
   USDCHF buy 0.1 lot margin: 100.0 USD
   EURJPY buy 0.1 lot margin: 109.91 USD
   GBPJPY buy 0.1 lot margin: 122.73 USD
```

另见

© 2000-2025, MetaQuotes Ltd.

order_calc_profit、order_check

© 2000-2025, MetaQuotes Ltd.

# order_calc_profit

返回指定交易操作的盈利（用账户货币表示）。

```
order_calc_profit(
   action,           // 订单类型 (ORDER_TYPE_BUY or ORDER_TYPE_SELL)
   symbol,           // 交易品种名称
   volume,           // 交易量
   price_open,       // 开盘价
   price_close       // 收盘价
   );
```

参数

*action*

[in] 订单类型可以获取两个ORDER_TYPE枚举值其中一个：ORDER_TYPE_BUY或ORDER_TYPE_SELL。所需的未命名参数。

*symbol*

[in] 交易品种名称。所需的未命名参数。

*volume*

[in] 交易操作的交易量。所需的未命名参数。

*price_open*

[in] 开盘价。所需的未命名参数。

*price_close*

[in] 收盘价。所需的未命名参数。

返回值

如果成功返回真实值，否则返回None。可以使用last_error()获取错误信息。

注意

该函数允许评估当前账户和当前交易环境下的交易操作结果。该函数类似于OrderCalcProfit。

例如：

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
```

© 2000-2025, MetaQuotes Ltd.

```
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# arrange the symbol list
symbols = ("EURUSD","GBPUSD","USDJPY")
print("Symbols to check margin:", symbols)
# estimate profit for buying and selling
lot=1.0
distance=300
for symbol in symbols:
    symbol_info=mt5.symbol_info(symbol)
    if symbol_info is None:
        print(symbol,"not found, skipped")
        continue
    if not symbol_info.visible:
        print(symbol, "is not visible, trying to switch on")
        if not mt5.symbol_select(symbol,True):
            print("symbol_select({}}) failed, skipped",symbol)
            continue
    point=mt5.symbol_info(symbol).point
    symbol_tick=mt5.symbol_info_tick(symbol)
    ask=symbol_tick.ask
    bid=symbol_tick.bid
    buy_profit=mt5.order_calc_profit(mt5.ORDER_TYPE_BUY,symbol,lot,ask,ask+distance*po
    if buy_profit!=None:
        print("   buy {} {} lot: profit on {} points => {} {}".format(symbol,lot,dista
    else:
        print("order_calc_profit(ORDER_TYPE_BUY) failed, error code =",mt5.last_error
    sell_profit=mt5.order_calc_profit(mt5.ORDER_TYPE_SELL,symbol,lot,bid,bid-distance*
    if sell_profit!=None:
        print("   sell {} {} lots: profit on {} points => {} {}".format(symbol,lot,dis
    else:
        print("order_calc_profit(ORDER_TYPE_SELL) failed, error code =",mt5.last_erro
    print()

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currency: USD
Symbols to check margin: ('EURUSD', 'GBPUSD', 'USDJPY')
   buy EURUSD 1.0 lot: profit on 300 points => 300.0 USD
   sell EURUSD 1.0 lot: profit on 300 points => 300.0 USD

   buy GBPUSD 1.0 lot: profit on 300 points => 300.0 USD
   sell GBPUSD 1.0 lot: profit on 300 points => 300.0 USD
```

© 2000-2025, MetaQuotes Ltd.

```
buy USDJPY 1.0 lot: profit on 300 points => 276.54 USD
sell USDJPY 1.0 lot: profit on 300 points => 278.09 USD
```

另见

order_calc_margin、order_check

© 2000-2025, MetaQuotes Ltd.

# order_check

检查是否有足够的资金执行所需的交易操作。检查结果以MqlTradeCheckResult结构返回。

```
order_check(
   request       // 请求结构
   );
```

参数

*request*

[in] 描述所需交易操作的MqlTradeRequest类型结构。所需的未命名参数。下面描述了填写请求和枚举内容的示例。

返回值

检查结果以MqlTradeCheckResult结构返回。回答中的*request*字段包含传递到order_check()的交易请求的结构。

注意

成功发送请求不代表所请求的交易操作就可成功执行。order_check函数类似于OrderCheck。

TRADE_REQUEST_ACTIONS

| ID | 描述 |
|---|---|
| TRADE_ACTION_DEAL | 以指定参数（设置市价单）下现价交易订单。 |
| TRADE_ACTION_PENDING | 在指定条件下（挂单）下执行交易订单， |
| TRADE_ACTION_SLTP | 更改持仓止损和止盈 |
| TRADE_ACTION_MODIFY | 更改之前所下交易订单的参数 |
| TRADE_ACTION_REMOVE | 删除之前所下的挂单 |
| TRADE_ACTION_CLOSE_BY | 反向平仓 |

ORDER_TYPE_FILLING

| ID | 描述 |
|---|---|
| ORDER_FILLING_FOK | 这个执行规则意味着一笔订单只能按照指定的交易量执行。如果金融产品的必要数额目前在市场上不能满足，则该订单将不会被执行。所需交易量可以由几个可用报价组成。 |
| ORDER_FILLING_IOC | 在订单中规定的交易量范围内，以市场最大交易量执行交易的协议。如果不能完全成交，则将执行可用交易量的订单，并取消其余交易量。 |
| ORDER_FILLING_RETURN | 这个规则只适用于市价单（ORDER_TYPE_BUY和ORDER_TYPE_SELL）、limit和stop limit订单（ORDER_TYPE_BUY_LIMIT、ORDER_TYPE_SELL_LIMIT、ORDER_TYPE_BUY_STOP_LIMIT和 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| | ORDER_TYPE_SELL_STOP_LIMIT)以及市价执行和交易所执行模式的交易品种。如果部分成交，则剩余交易量的订单不会被取消，等待进一步处理。<br>在激活ORDER_TYPE_BUY_STOP_LIMIT和ORDER_TYPE_SELL_STOP_LIMIT订单期间，将创建一个对应的限价单<br>ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT，并使用ORDER_FILLING_RETURN类型。 |

ORDER_TYPE_TIME

| ID | 描述 |
| --- | --- |
| ORDER_TIME_GTC | 订单将一直保留在队列中，直到手动取消 |
| ORDER_TIME_DAY | 订单仅在当前交易日有效 |
| ORDER_TIME_SPECIFIED | 订单在指定日期之前有效 |
| ORDER_TIME_SPECIFIED_DAY | 订单在指定日23:59:59之前有效。如果此时间不在交易时段内，到期将在最近的交易时间内处理。 |

例如：

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# 准备请求结构
symbol="USDJPY"
symbol_info = mt5.symbol_info(symbol)
if symbol_info is None:
    print(symbol, "not found, can not call order_check()")
    mt5.shutdown()
    quit()

# 如果市场报价中没有此交易品种，请添加
if not symbol_info.visible:
    print(symbol, "is not visible, trying to switch on")
    if not mt5.symbol_select(symbol,True):
        print("symbol_select({}) failed, exit",symbol)
        mt5.shutdown()
        quit()
```

© 2000-2025, MetaQuotes Ltd.

```
# prepare the request
point=mt5.symbol_info(symbol).point
request = {
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": 1.0,
    "type": mt5.ORDER_TYPE_BUY,
    "price": mt5.symbol_info_tick(symbol).ask,
    "sl": mt5.symbol_info_tick(symbol).ask-100*point,
    "tp": mt5.symbol_info_tick(symbol).ask+100*point,
    "deviation": 10,
    "magic": 234000,
    "comment": "python script",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}

# 执行检查并显示结果'按原状'
result = mt5.order_check(request)
print(result);
# request the result as a dictionary and display it element by element
result_dict=result._asdict()
for field in result_dict.keys():
    print("   {}={}".format(field,result_dict[field]))
    # if this is a trading request structure, display it element by element as well
    if field=="request":
        traderequest_dict=result_dict[field]._asdict()
        for tradereq_filed in traderequest_dict:
            print("       traderequest: {}={}".format(tradereq_filed,traderequest_dict

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currency: USD
   retcode=0
   balance=101300.53
   equity=68319.53
   profit=-32981.0
   margin=51193.67
   margin_free=17125.86
   margin_level=133.45308121101692
   comment=Done
   request=TradeRequest(action=1, magic=234000, order=0, symbol='USDJPY', volume=1.0,
       traderequest: action=1
       traderequest: magic=234000
       traderequest: order=0
       traderequest: symbol=USDJPY
       traderequest: volume=1.0
```

© 2000-2025, MetaQuotes Ltd.

```
        traderequest: price=108.081
        traderequest: stoplimit=0.0
        traderequest: sl=107.98100000000001
        traderequest: tp=108.181
        traderequest: deviation=10
        traderequest: type=0
        traderequest: type_filling=2
        traderequest: type_time=0
        traderequest: expiration=0
        traderequest: comment=python script
        traderequest: position=0
        traderequest: position_by=0
```

另见

order_send、OrderCheck、交易操作类型、交易请求结构、交易请求结构的检查结果、交易请求结构的结果

© 2000-2025, MetaQuotes Ltd.

# order_send

从程序端向交易服务器发送请求来执行交易操作。该函数类似于OrderSend。

```
order_send(
   request       // 请求结构
   );
```

参数

*request*

[in] 描述所需交易操作的MqlTradeRequest类型结构。所需的未命名参数。下面描述了填写请求和枚举内容的示例。

返回值

执行结果以MqlTradeResult结构返回。回答中的*request*字段包含传递到order_send()的交易请求的结构。可使用last_error()获取错误信息。

MqlTradeRequest交易请求结构

| 字段 | 描述 |
|---|---|
| action | 交易请求类型。该值可以是TRADE_REQUEST_ACTIONS其中一个枚举值 |
| 幻数 | EA ID。可以安排交易订单的分析处理。每个EA交易都可以在发送交易请求时设置一个专有ID |
| 订单 | 订单单号。更改挂单所需 |
| symbol | 要下单的交易品种名称。更改订单和平仓时不需要 |
| volume | 请求的交易量（以手数表示）交易时的真实交易量取决于订单执行类型。 |
| price | 执行订单的价格。对于具有TRADE_ACTION_DEAL类型的“市价执行”(SYMBOL_TRADE_EXECUTION_MARKET)类型的交易品种的市价单，不设置价格 |
| 止损限价 | 当价格达到'price'值时将设置Limit挂单（此条件是强制性的）。在此之前，挂单不会传递到交易系统中 |
| sl | 当价格向不利的方向移动时，激活止损单 |
| tp | 当价格向有利的方向移动时，激活止盈单 |
| 偏差 | 请求价格的最大可接受偏差，在points中指定 |
| 类型 | 订单类型。该值可以是ORDER_TYPE其中一个枚举值 |
| type_filling | 订单成交类型。该值可以是ORDER_TYPE_FILLING其中一个值 |
| type_time | 订单到期类型。该值可以是ORDER_TYPE_TIME其中一个值 |
| 到期 | 挂单到期时间（用于TIME_SPECIFIED类型订单） |
| 评论 | 订单注释 |

© 2000-2025, MetaQuotes Ltd.

| 字段 | 描述 |
| --- | --- |
| 持仓 | 持仓单号。当更改和关闭持仓时为清楚识别而填写。通常，它与持仓的订单单号相同。 |
| position_by | 反向持仓单号。它用于通过反向持仓平仓时（反向打开一个同名的交易品种）。 |

注意

交易请求在交易服务器上通过多个验证阶段。首先，检查所有必要*request*字段的有效性。如果没有错误，服务器接受订单以进一步处理。有关执行交易操作的详细信息，请参阅OrderSend函数的描述。

例如：

```
import time
import MetaTrader5 as mt5

# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ", mt5.__author__)
print("MetaTrader5 package version: ", mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 准备买入请求结构
symbol = "USDJPY"
symbol_info = mt5.symbol_info(symbol)
if symbol_info is None:
    print(symbol, "not found, can not call order_check()")
    mt5.shutdown()
    quit()

# 如果市场报价中没有此交易品种，请添加
if not symbol_info.visible:
    print(symbol, "is not visible, trying to switch on")
    if not mt5.symbol_select(symbol,True):
        print("symbol_select({}) failed, exit",symbol)
        mt5.shutdown()
        quit()

lot = 0.1
point = mt5.symbol_info(symbol).point
price = mt5.symbol_info_tick(symbol).ask
deviation = 20
request = {
    "action": mt5.TRADE_ACTION_DEAL,
```

© 2000-2025, MetaQuotes Ltd.

```
    "symbol": symbol,
    "volume": lot,
    "type": mt5.ORDER_TYPE_BUY,
    "price": price,
    "sl": price - 100 * point,
    "tp": price + 100 * point,
    "deviation": deviation,
    "magic": 234000,
    "comment": "python script open",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}

# 发送交易请求
result = mt5.order_send(request)
# 检查执行结果
print("1. order_send(): by {} {} lots at {} with deviation={} points".format(symbol,lo
if result.retcode != mt5.TRADE_RETCODE_DONE:
    print("2. order_send failed, retcode={}".format(result.retcode))
   # 请求词典结果并逐个元素显示
    result_dict=result._asdict()
    for field in result_dict.keys():
        print("   {}={}".format(field,result_dict[field]))
        # if this is a trading request structure, display it element by element as wel
        if field=="request":
            traderequest_dict=result_dict[field]._asdict()
            for tradereq_filed in traderequest_dict:
                print("       traderequest: {}={}".format(tradereq_filed,traderequest_
    print("shutdown() and quit")
    mt5.shutdown()
    quit()

print("2. order_send done, ", result)
print("   opened position with POSITION_TICKET={}".format(result.order))
print("   sleep 2 seconds before closing position #{}".format(result.order))
time.sleep(2)
# 创建一个关闭请求
position_id=result.order
price=mt5.symbol_info_tick(symbol).bid
deviation=20
request={
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": lot,
    "type": mt5.ORDER_TYPE_SELL,
    "position": position_id,
    "price": price,
    "deviation": deviation,
    "magic": 234000,
```

© 2000-2025, MetaQuotes Ltd.

```
    "comment": "python script close",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}
# 发送交易请求
result=mt5.order_send(request)
# 检查执行结果
print("3. close position #{}: sell {} {} lots at {} with deviation={} points".format(p
if result.retcode != mt5.TRADE_RETCODE_DONE:
    print("4. order_send failed, retcode={}".format(result.retcode))
    print("   result",result)
其他:
    print("4. position #{} closed, {}".format(position_id,result))
   # 请求词典结果并逐个元素显示
    result_dict=result._asdict()
    for field in result_dict.keys():
        print("   {}={}".format(field,result_dict[field]))
        # if this is a trading request structure, display it element by element as wel
        if field=="request":
            traderequest_dict=result_dict[field]._asdict()
            for tradereq_filed in traderequest_dict:
                print("       traderequest: {}={}".format(tradereq_filed,traderequest_

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

结果

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29
1. order_send(): by USDJPY 0.1 lots at 108.023 with deviation=20 points
2. order_send done,  OrderSendResult(retcode=10009, deal=535084512, order=557416535, v
   opened position with POSITION_TICKET=557416535
   sleep 2 seconds before closing position #557416535
3. close position #557416535: sell USDJPY 0.1 lots at 108.018 with deviation=20 points
4. position #557416535 closed, OrderSendResult(retcode=10009, deal=535084631, order=55
   retcode=10009
   deal=535084631
   order=557416654
   volume=0.1
   price=108.015
   bid=108.015
   ask=108.02
   comment=Request executed
   request_id=55
   retcode_external=0
   request=TradeRequest(action=1, magic=234000, order=0, symbol='USDJPY', volume=0.1,
       traderequest: action=1
       traderequest: magic=234000
       traderequest: order=0
```

© 2000-2025, MetaQuotes Ltd.

```
        traderequest: symbol=USDJPY
        traderequest: volume=0.1
        traderequest: price=108.018
        traderequest: stoplimit=0.0
        traderequest: sl=0.0
        traderequest: tp=0.0
        traderequest: deviation=20
        traderequest: type=1
        traderequest: type_filling=2
        traderequest: type_time=0
        traderequest: expiration=0
        traderequest: comment=python script close
        traderequest: position=557416535
        traderequest: position_by=0
```

另见

order_check、OrderSend、交易操作类型、交易请求结构、交易请求结构的检查结果、交易请求结构的结果

© 2000-2025, MetaQuotes Ltd.

# positions_total

获取未结持仓的数量。

```
positions_total()
```

返回值

整数值。

注意

该函数类似于PositionsTotal。

例如：

```
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 检查是否存在未结持仓
positions_total=mt5.positions_total()
if positions_total>0:
    print("Total positions=",positions_total)
其他:
    print("Positions not found")

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

另见

positions_get、orders_total

© 2000-2025, MetaQuotes Ltd.

# positions_get

获取可通过交易品种或单号筛选的未结持仓。共有三个调用选项。

无参数调用。返回所有交易品种的未结持仓。

```
positions_get()
```

调用指定应接收的交易品种未结持仓。

```
positions_get(
   symbol="SYMBOL"       // 交易品种名称
)
```

调用指定应接收的交易品种组未结持仓。

```
positions_get(
   group="GROUP"         // 根据交易品种选择持仓的过滤器
)
```

调用指定持仓单号。

```
positions_get(
   ticket=TICKET         // 单号
)
```

参数

*symbol="SYMBOL"*

[in] 交易品种名称。可选的命名参数。如果指定交易品种，则忽略*ticket*参数。

*group="GROUP"*

[in] 用于跑了一组必要交易品种的过滤器。可选的命名参数。如果指定了组，则函数只返回满足交易品种名称指定条件的持仓。

*ticket=TICKET*

[in] 持仓单号 (POSITION_TICKET)。可选的命名参数。

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数允许在一个类似于PositionsTotal和PositionSelect的串联调用中接收所有未结持仓。

*group*参数可以包含多个逗号分隔条件。条件可以使用'*'设置为掩码。逻辑否定字符'!'可以用来表示排除条件。所有条件依序应用，这意味着应该先指定包含到组中的条件，然后再指定排除条件。例如，*group*="*, !EUR"意味着应该先选择所有交易品种的持仓，然后排除交易品种名称中包含"EUR"的持仓。

例如：

```
import MetaTrader5 as mt5
import pandas as pd
```

© 2000-2025, MetaQuotes Ltd.

```
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取USDCHF的未结持仓
positions=mt5.positions_get(symbol="USDCHF")
if positions==None:
    print("No positions on USDCHF, error code={}".format(mt5.last_error()))
elif len(positions)>0:
    print("Total positions on USDCHF =",len(positions))
    # display all open positions
    for position in positions:
        print(position)

# 获取名称包含"*USD*"的交易品种的持仓列表
usd_positions=mt5.positions_get(group="*USD*")
if usd_positions==None:
    print("No positions with group=\"*USD*\", error code={}".format(mt5.last_error()))
elif len(usd_positions)>0:
    print("positions_get(group=\"*USD*\")={}".format(len(usd_positions)))
    # display these positions as a table using pandas.DataFrame
    df=pd.DataFrame(list(usd_positions),columns=usd_positions[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    df.drop(['time_update', 'time_msc', 'time_update_msc', 'external_id'], axis=1, inp
    print(df)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

positions_get(group="*USD*")=5
      ticket                time  type  magic  identifier  reason  volume  price_open
0  548297723 2020-03-18 15:00:55     1      0   548297723       3    0.01     1.09301
1  548655158 2020-03-18 20:31:26     0      0   548655158       3    0.01     1.08676
2  548663803 2020-03-18 20:40:04     0      0   548663803       3    0.01     1.08640
3  548847168 2020-03-19 01:10:05     0      0   548847168       3    0.01     1.09545
4  548847194 2020-03-19 01:10:07     0      0   548847194       3    0.02     1.09536
```

© 2000-2025, MetaQuotes Ltd.

另见

positions_total、orders_get

© 2000-2025, MetaQuotes Ltd.

# history_deals_total

获取交易历史中指定时间间隔内的交易数量。

```
history_deals_total(
   date_from,     // 请求交易的开始日期
   date_to        // 请求交易的结束日期
   )
```

参数

*date_from*

[in] 请求交易的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*date_to*

[in] 请求交易的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

返回值

整数值。

注意

该函数类似于HistoryDealsTotal。

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取历史中的交易数量
from_date=datetime(2020,1,1)
to_date=datetime.now()
deals=mt5.history_deals_total(from_date, to_date)
if deals>0:
    print("Total deals=",deals)
其他:
    print("Deals not found in history")

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

© 2000-2025, MetaQuotes Ltd.

另见

history_deals_get、history_orders_total

© 2000-2025, MetaQuotes Ltd.

# history_deals_get

获取可通过单号或持仓筛选的指定时间间隔内的交易历史中的交易。

调用指定时间间隔。返回指定时间间隔内的所有交易。

```
history_deals_get(
   date_from,            // 请求交易的开始日期
   date_to,              // 请求交易的结束日期
   group="GROUP"         // 为交易品种选择交易的过滤器
   )
```

调用指定订单单号。返回在DEAL_ORDER属性中具有指定订单单号的所有交易。

```
history_deals_get(
   ticket=TICKET         //订单单号
)
```

调用指定持仓单号。返回在DEAL_POSITION_ID属性中具有指定持仓单号的所有交易。

```
history_deals_get(
   position=POSITION     //持仓单号
)
```

参数

*date_from*

[in] 请求订单的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。首先指定所需的未命名参数。

*date_to*

[in] 请求订单的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。其次指定所需的未命名参数。

*group="GROUP"*

[in] 用于跑了一组必要交易品种的过滤器。可选的命名参数。如果指定了组，则函数只返回满足交易品种名称指定条件的交易。

*ticket=TICKET*

[in] 应接收所有交易的订单单号（存储在DEAL_ORDER）。可选参数。如果没有指定，则不应用过滤器。

*position=POSITION*

[in] 应接收所有交易的持仓单号（存储在DEAL_POSITION_ID）。可选参数。如果没有指定，则不应用过滤器。

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数允许在一个类似于HistoryDealsTotal和HistoryDealSelect的串联调用中接收指定时间段内的所有历史交易。

© 2000-2025, MetaQuotes Ltd.

*group*参数可以根据交易品种对交易进行分类。'*'可以用在字符串的开头和结尾。

*group*参数可以包含多个逗号分隔条件。条件可以使用'*'设置为掩码。逻辑否定字符'!'可以用来表示排除条件。所有条件依序应用，这意味着应该先指定包含到组中的条件，然后再指定排除条件。例如，*group*="*, !EUR"意味着应该先选择所有交易品种的交易，然后排除交易品种名称中包含"EUR"的交易。

例如：

```
import MetaTrader5 as mt5
from datetime import datetime
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取历史中的交易数量
from_date=datetime(2020,1,1)
to_date=datetime.now()
# 获取指定时间间隔内且名称包含"GBP"的交易品种的交易
deals=mt5.history_deals_get(from_date, to_date, group="*GBP*")
if deals==None:
    print("No deals with group=\"*USD*\", error code={}".format(mt5.last_error()))
elif len(deals)> 0:
    print("history_deals_get({}, {}, group=\"*GBP*\")={}".format(from_date,to_date,len

# 获取名称中既不包含"EUR"也不包含"GBP"的交易品种的交易
deals = mt5.history_deals_get(from_date, to_date, group="*,!*EUR*,!*GBP*")
if deals == None:
    print("No deals, error code={}".format(mt5.last_error()))
elif len(deals) > 0:
    print("history_deals_get(from_date, to_date, group=\"*,!*EUR*,!*GBP*\") =", len(de
    # display all obtained deals 'as is'
    for deal in deals:
        print("  ",deal)
    print()
    # display these deals as a table using pandas.DataFrame
    df=pd.DataFrame(list(deals),columns=deals[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    print(df)
print("")

# 获取#530218319持仓相关的所有交易
```

© 2000-2025, MetaQuotes Ltd.

```
position_id=530218319
position_deals = mt5.history_deals_get(position=position_id)
if position_deals == None:
    print("No deals with position #{}".format(position_id))
    print("error code =", mt5.last_error())
elif len(position_deals) > 0:
    print("Deals with position id #{}: {}".format(position_id, len(position_deals)))
    # display these deals as a table using pandas.DataFrame
    df=pd.DataFrame(list(position_deals),columns=position_deals[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    print(df)

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

结果：

```
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

history_deals_get(from_date, to_date, group="*GBP*") = 14

history_deals_get(from_date, to_date, group="*,!*EUR*,!*GBP*") = 7
   TradeDeal(ticket=506966741, order=0, time=1582202125, time_msc=1582202125419, type=
   TradeDeal(ticket=507962919, order=530218319, time=1582303777, time_msc=158230377758
   TradeDeal(ticket=513149059, order=535548147, time=1583176242, time_msc=158317624226
   TradeDeal(ticket=516943494, order=539349382, time=1583510003, time_msc=158351000389
   TradeDeal(ticket=516943915, order=539349802, time=1583510025, time_msc=158351002505
   TradeDeal(ticket=517139682, order=539557870, time=1583520201, time_msc=158352020122
   TradeDeal(ticket=517139716, order=539557909, time=1583520202, time_msc=158352020297

      ticket      order                time       time_msc  type  entry  magic  positi
0  506966741          0 2020-02-20 12:35:25  1582202125419     2      0      0
1  507962919  530218319 2020-02-21 16:49:37  1582303777582     0      0      0    5302
2  513149059  535548147 2020-03-02 19:10:42  1583176242265     1      1      0    5302
3  516943494  539349382 2020-03-06 15:53:23  1583510003895     1      0      0    5393
4  516943915  539349802 2020-03-06 15:53:45  1583510025054     0      0      0    5393
5  517139682  539557870 2020-03-06 18:43:21  1583520201227     0      1      0    5393
6  517139716  539557909 2020-03-06 18:43:22  1583520202971     1      1      0    5393

Deals with position id #530218319: 2
      ticket      order                time       time_msc  type  entry  magic  positi
0  507962919  530218319 2020-02-21 16:49:37  1582303777582     0      0      0    5302
1  513149059  535548147 2020-03-02 19:10:42  1583176242265     1      1      0    5302
```

另见

history_deals_total、history_orders_get

© 2000-2025, MetaQuotes Ltd.

# history_orders_total

获取交易历史中指定时间间隔内的订单数量。

```
history_orders_total(
   date_from,     // 请求订单的开始日期
   date_to        //请求订单的结束日期
   )
```

参数

*date_from*

[in] 请求订单的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

*date_to*

[in] 请求订单的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。所需的未命名参数。

返回值

整数值。

注意

The function is similar to HistoryOrdersTotal.

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取历史中的订单数量
from_date=datetime(2020,1,1)
to_date=datetime.now()
history_orders=mt5.history_orders_total(from_date, datetime.now())
if history_orders>0:
    print("Total history orders=",history_orders)
其他:
    print("Orders not found in history")

# 断开与MetaTrader 5程序端的连接
mt5.shutdown()
```

© 2000-2025, MetaQuotes Ltd.

另见

history_orders_get、history_deals_total

© 2000-2025, MetaQuotes Ltd.

# history_orders_get

获取可通过单号或持仓筛选的交易历史中的订单。共有三个调用选项。

调用指定时间间隔。返回指定时间间隔内的所有订单。

```
history_orders_get(
   date_from,            // 请求订单的开始日期
   date_to,              // 请求订单的结束日期
   group="GROUP"         // 根据交易品种选择订单的过滤器
   )
```

调用指定订单单号。返回带有指定单号的订单。

```
history_orders_get(
   ticket=TICKET         //订单单号
)
```

调用指定持仓单号。返回在ORDER_POSITION_ID属性中具有指定持仓单号的所有订单。

```
history_orders_get(
   position=POSITION     //持仓单号
)
```

参数

*date_from*

[in] 请求订单的开始日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。首先指定所需的未命名参数。

*date_to*

[in] 请求订单的结束日期。通过'datetime'对象或根据1970.01.01以来过去的秒数设置。其次指定所需的未命名参数。

*group="GROUP"*

[in] 用于跑了一组必要交易品种的过滤器。可选的命名参数。如果指定了组，则函数只返回满足交易品种名称指定条件的订单。

*ticket=TICKET*

[in] 应该接收的订单单号。可选参数。如果没有指定，则不应用过滤器。

*position=POSITION*

[in] 应接收所有交易的持仓单号（存储在ORDER_POSITION_ID）。可选参数。如果没有指定，则不应用过滤器。

返回值

以指定元组结构(namedtuple)的形式返回信息。错误情况下返回None。可使用last_error()获取错误信息。

注意

该函数允许在一个类似于HistoryOrdersTotal和HistoryOrderSelect的串联调用中接收指定时间段内的所有历史订单。

© 2000-2025, MetaQuotes Ltd.

*group*参数可以包含多个逗号分隔条件。条件可以使用'*'设置为掩码。逻辑否定字符'!'可以用来表示排除条件。所有条件依序应用，这意味着应该先指定包含到组中的条件，然后再指定排除条件。例如，*group*="*, !EUR"意味着应该先选择所有交易品种的交易，然后排除交易品种名称中包含"EUR"的交易。

例如：

```
from datetime import datetime
import MetaTrader5 as mt5
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# 显示有关MetaTrader 5程序包的数据
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# 建立与MetaTrader 5程序端的连接
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# 获取历史中的订单数量
from_date=datetime(2020,1,1)
to_date=datetime.now()
history_orders=mt5.history_orders_get(from_date, to_date, group="*GBP*")
if history_orders==None:
    print("No history orders with group=\"*GBP*\", error code={}".format(mt5.last_erro
elif len(history_orders)>0:
    print("history_orders_get({}, {}, group=\"*GBP*\")={}".format(from_date,to_date,le
print()

# 根据持仓单号显示所有历史订单
position_id=530218319
position_history_orders=mt5.history_orders_get(position=position_id)
if position_history_orders==None:
    print("No orders with position #{}".format(position_id))
    print("error code =",mt5.last_error())
elif len(position_history_orders)>0:
    print("Total history orders on position #{}: {}".format(position_id,len(position_h
   # 显示带有指定持仓单号的所有历史订单
    for position_order in position_history_orders:
        print(position_order)
    print()
    # display these orders as a table using pandas.DataFrame
    df=pd.DataFrame(list(position_history_orders),columns=position_history_orders[0]._
    df.drop(['time_expiration','type_time','state','position_by_id','reason','volume_
    df['time_setup'] = pd.to_datetime(df['time_setup'], unit='s')
    df['time_done'] = pd.to_datetime(df['time_done'], unit='s')
    print(df)
```

© 2000-2025, MetaQuotes Ltd.

```
# 断开与MetaTrader 5程序端的连接
mt5.shutdown()

结果:
MetaTrader5程序包作者:MetaQuotes Software Corp.
MetaTrader5程序包版本:5.0.29

history_orders_get(2020-01-01 00:00:00, 2020-03-25 17:17:32.058795, group="*GBP*")=14

Total history orders on position #530218319: 2
TradeOrder(ticket=530218319, time_setup=1582282114, time_setup_msc=1582282114681, time
TradeOrder(ticket=535548147, time_setup=1583176242, time_setup_msc=1583176242265, time

      ticket          time_setup  time_setup_msc           time_done  time_done_msc  t
0  530218319 2020-02-21 10:48:34   1582282114681 2020-02-21 16:49:37  1582303777582
1  535548147 2020-03-02 19:10:42   1583176242265 2020-03-02 19:10:42  1583176242265
```

另 见

history_deals_total、history_deals_get

© 2000-2025, MetaQuotes Ltd.

# 机器学习中的ONNX模型

ONNX (Open NeuralNetwork Exchange)是机器学习模型的开源格式。这个项目有几个主要优势：

- ONNX得到Microsoft、Facebook、Amazon等大公司的支持。
- 它的开放格式支持不同机器学习工具包之间的格式转换，而Microsoft (微软) ONNXMLTools允许将模型转换为ONNX格式。
- 如果传递的参数类型与模型不匹配，MQL5会为模型输入和输出提供自动数据类型转换。
- ONNX模型可以使用各种机器学习工具创建。目前在Caffe2、Microsoft Cognitive Toolkit、MXNet、PyTorch和OpenCV中得到支持。还可以使用其他流行框架和程序库的接口。
- 借助MQL5语言，您可以在交易策略中实施ONNX模型，并将其与MetaTrader 5平台的所有优势结合使用，从而在金融市场中实现高效操作。
- 在为实时交易调整模型之前，您可以在策略测试中根据历史数据测试模型行为，而无需使用第三方工具。

MQL5为使用ONNX提供以下函数：

| 函数 | 操作 |
| --- | --- |
| OnnxCreate | 创建ONNX会话，从 *.onnx文件中加载模型 |
| OnnxCreateFromBuffer | 创建ONNX会话，从数据数组加载模型 |
| OnnxRelease | 关闭ONNX会话 |
| OnnxRun | 运行ONNX模型 |
| OnnxGetInputCount | 获取ONNX模型中的输入数 |
| OnnxGetOutputCount | 获取ONNX模型中的输出数 |
| OnnxGetInputName | 通过索引获取模型输入名称 |
| OnnxGetOutputName | 通过索引获取模型输出名称 |
| OnnxGetInputTypeInfo | 从模型获取输入类型的描述 |
| OnnxGetOutputTypeInfo | 从模型获取输出类型的描述 |
| OnnxSetInputShape | 通过索引设置模型输入数据的形状 |
| OnnxSetOutputShape | 通过索引设置模型输出数据的形状 |

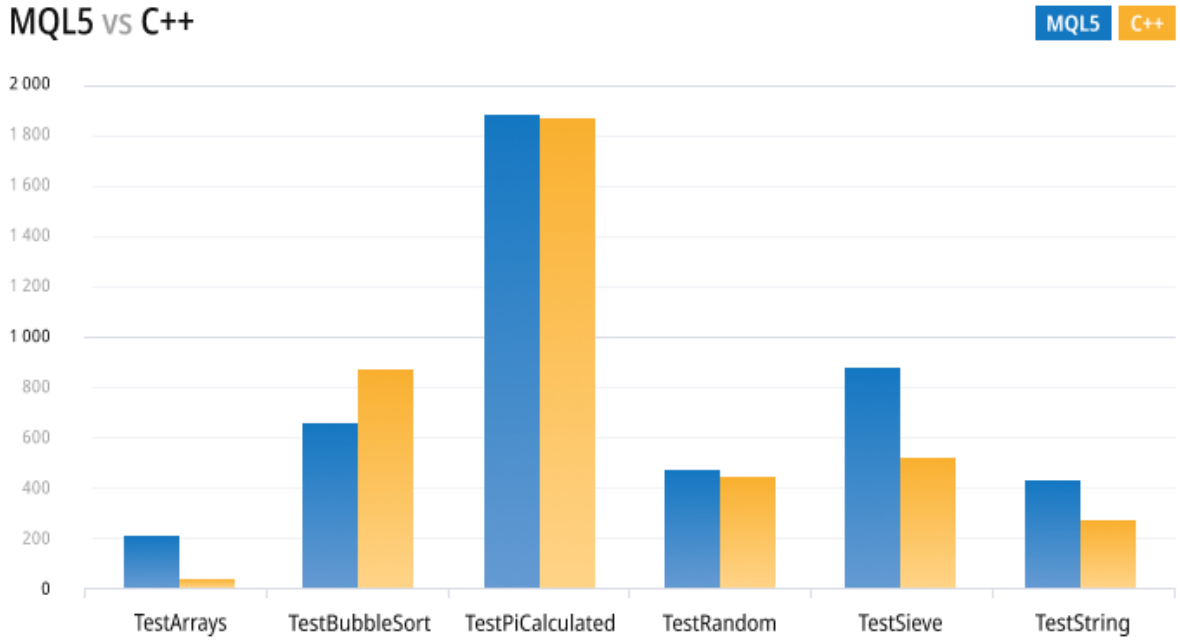
© 2000-2025, MetaQuotes Ltd.

# MQL5中的ONNX支持

ONNX是一种用于表示机器学习模型的开放格式构建。该标准定义了一组通用的运算符和通用的文件格式，使开发人员能够使用不同框架、工具、运行时间和编译器的模型。

因此，ONNX开放格式允许您在不同的平台和机器学习工具包之间接收和传输机器学习模型。ONNX以MQL5语言实施，使AI开发人员能够在MetaTrader 5平台提供的高性能执行环境中运行创建的模型。

MQL5执行速度与C++应用程序的执行速度相当。MQL5和C++标准测试的执行结果证明了这一点。柱形图越低，执行所花费的时间越少（以毫秒为单位），结果越好。测试执行在Windows 10(build 17763) x64、Xeon E5-2630 v4 @ 2.20GHz、内存：65457 Mb。




新异步交易操作和本地ONNX支持为您提供了以前只有少数专业AI开发人员和机构交易者才能获得的新机会。MQL5中的ONNX支持使交易者能够在他们喜欢的开发环境中训练金融市场交易模型，然后以低网络成本、高订单簿更新速度和异步订单提交进行交易。

目前，ONNX正在由Microsoft（微软）、Facebook、Amazon（亚马逊）等合作伙伴公司共同开发和维护，为这个开放项目的进一步发展提供了保障。

© 2000-2025, MetaQuotes Ltd.

# Format Conversion（格式转换）

ONNX是一个开放格式，允许使用来自不同机器学习工具包的模型。许多框架都支持该格式，包括Chainer、Caffee2和PyTorch。

将模型转换为ONNX格式的最受欢迎的工具之一是微软ONNXMLTools。

ONNXMLTools安装和使用说明可在 GitHub repo中获得。目前支持以下工具包：

- Keras (a wrapper of keras2onnx converter)
- Tensorflow (a wrapper of tf2onnx converter)
- scikit-learn (a wrapper of skl2onnx converter)
- Apple Core ML
- Spark ML (experimental)
- LightGBM
- libscm;
- XGBoost;
- H2O
- CatBoost

ONNXMLTools可以轻松安装。有关安装细节和模型转换示例，请参阅项目页 https://github.com/onnx/onnxmltools#install。

© 2000-2025, MetaQuotes Ltd.

# 运行ONNX模型时自动转换输入和输出值

MQL5中当前ONNX版本仅支持输入/输出值的张量。张量是包含以下数据类型元素的数据数组：

| ONNX类型 | 对应于MQL5类型 |
| --- | --- |
| ONNX_DATA_TYPE_BOOL | 布尔型 |
| ONNX_DATA_TYPE_FLOAT | 浮点型 |
| ONNX_DATA_TYPE_UINT8 | 无字符型(uchar)< |
| ONNX_DATA_TYPE_INT8 | 字符型(char) |
| ONNX_DATA_TYPE_UINT16 | 无符短型(ushort) |
| ONNX_DATA_TYPE_INT16 | 短整型(short) |
| ONNX_DATA_TYPE_INT32 | 整型(int) |
| ONNX_DATA_TYPE_INT64 | 长整型(long) |
| ONNX_DATA_TYPE_FLOAT16 | – |
| ONNX_DATA_TYPE_DOUBLE | 双精度 |
| ONNX_DATA_TYPE_UINT32 | 无符号整型(uint) |
| ONNX_DATA_TYPE_UINT64 | 无符长型(ulong) |
| ONNX_DATA_TYPE_COMPLEX64 | – |
| ONNX_DATA_TYPE_COMPLEX128 | 复杂型 |
| ONNX_DATA_TYPE_BFLOAT16 | – |
| ONNX_DATA_TYPE_STRING | – |

只有数组、向量和矩阵（我们将它们称为**数据**）可作为输入/输出值进入ONNX模型。

如果参数类型与ONNX模型的参数类型不匹配，并且在没有指定ONNX_NO_CONVERSION标识的情况下调用OnnxRun，将应用自动数据转换。自动转换意味着在运行ONNX模型之前，用户**数据**将通过相关转换被复制到ONNX张量中。

当ONNX模型在没有自动转换的情况下运行时，将使用**数据**计算模型，而无需任何额外的复制。

> **重要！** 自动转换不控制过载（截断），因此您应该仔细监控输入到ONNX模型中的数据和数据类型。

自动转换支持以下ONNX类型：

- ONNX_DATA_TYPE_BOOL
- ONNX_DATA_TYPE_FLOAT
- ONNX_DATA_TYPE_UINT8
- ONNX_DATA_TYPE_INT8
- ONNX_DATA_TYPE_UINT16

© 2000-2025, MetaQuotes Ltd.

- ONNX_DATA_TYPE_INT16
- ONNX_DATA_TYPE_INT32
- ONNX_DATA_TYPE_INT64
- ONNX_DATA_TYPE_FLOAT16
- ONNX_DATA_TYPE_DOUBLE
- ONNX_DATA_TYPE_UINT32
- ONNX_DATA_TYPE_UINT64
- ONNX_DATA_TYPE_COMPLEX64
- ONNX_DATA_TYPE_COMPLEX128

不支持的类型：

- ONNX_DATA_TYPE_BFLOAT16
- ONNX_DATA_TYPE_STRING

## 张量类型的自动转换规则

如果MQL5类型未包含在模型支持的类型列表中，运行ONNX模型将返回ERR_ONNX_NOT_SUPPORTED错误（错误代码5802）。

注意：自动转换时，color类型被处理为uint，而datetime被处理为long。

输入值的自动转换

| ONNX类型（张量项目类型） | 自动转换支持的MQL5类型 |
| --- | --- |
| ONNX_DATA_TYPE_BOOL | bool, char, uchar, short, ushort, int, color, uint, datetime, long, folat, double, complex<br><br>在转换过程中，通过与0的简单比较来检查Data元素 |
| ONNX_DATA_TYPE_FLOAT16 | float, double |
| ONNX_DATA_TYPE_FLOAT | char, uchar, short, ushort, int, color, uint, datetime, long, ulong, float, double |
| ONNX_DATA_TYPE_UINT8 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT8 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_UINT16 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT16 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT32 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT64 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_DOUBLE | 请见ONNX_DATA_TYPE_FLOAT |

© 2000-2025, MetaQuotes Ltd.

| ONNX类型（张量项目类型） | 自动转换支持的MQL5类型 |
| --- | --- |
| ONNX_DATA_TYPE_UINT32 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_UINT64 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_COMPLEX64 | 复杂型 |
| ONNX_DATA_TYPE_COMPLEX128 | 复杂型 |

输出值的自动转换

| ONNX类型（张量项目类型） | 自动转换支持的MQL5类型 |
| --- | --- |
| ONNX_DATA_TYPE_BOOL | bool, char, uchar, short, ushort, int, color, uint, datetime, long, folat, double, complex<br><br>如果张量元素为零，则Data元素设为0；否则，该值为1 |
| ONNX_DATA_TYPE_FLOAT16 | float, double |
| ONNX_DATA_TYPE_FLOAT | char, uchar, short, ushort, int, color, uint, datetime, long, ulong, float, double |
| ONNX_DATA_TYPE_UINT8 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT8 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_UINT16 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT16 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT32 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_INT64 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_DOUBLE | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_UINT32 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_UINT64 | 请见ONNX_DATA_TYPE_FLOAT |
| ONNX_DATA_TYPE_COMPLEX64 | 复杂型 |
| ONNX_DATA_TYPE_COMPLEX128 | 复杂型 |

另见

类型转换

© 2000-2025, MetaQuotes Ltd.

# 创建模型

有多种方法可以获得ONNX格式的现有模型。受欢迎的ONNX Model Zoo程序库包含针对不同任务类型的几个预训练ONNX模型。这个集合的优势是每个模型的笔记本都包含训练数据集的链接和对描述模型体系结构的原始论文的引用。

大部分机器学习框架使用Python。要对Python安装ONNX runtime，请在以下命令中选择一个：

```
pip install onnxruntime        # CPU build
pip install onnxruntime-gpu    # GPU build
```

要在Python中调用ONNX runtime，请使用以下命令

```
import onnxruntime
session = onnxruntime.InferenceSession("path to model")
```

对于模型输入和输出，请检查相关模型的文档。您还可以使用可视化工具来查看模型，例如Netron或WinML Dashboard。在ONNX runtime中，您还可以查询模型的元数据及其输入和输出：

```
results = session.run(["output1", "output2"], {
                      "input1": indata1, "input2": indata2})
results = session.run([], {"input1": indata1, "input2": indata2})
```

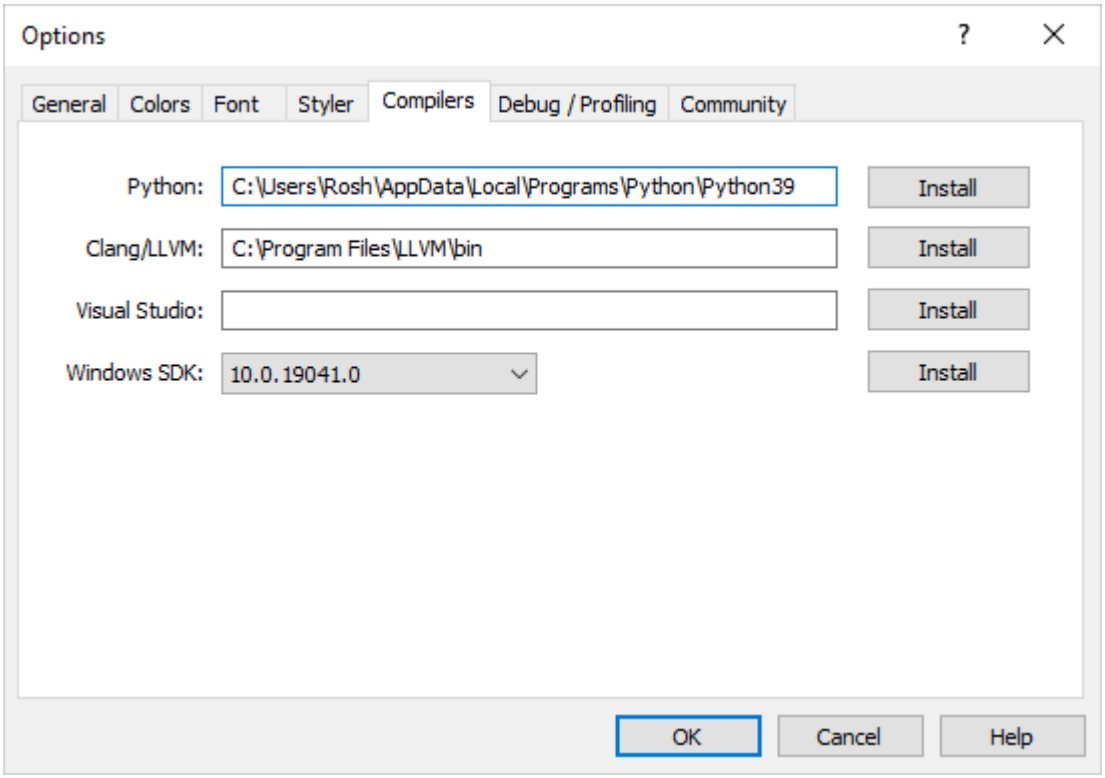
您可以使用Python直接在MetaTrader 5程序端或MetaEditor中创建ONNX模型。

## MetaTrader 5中的Python

MetaTrader 5为Python脚本提供了即时可用的支持。若要启用这些操作，程序端开发人员提供了MetaTrader5 Python模块：https://pypi.org/project/MetaTrader5。

在MetaEditor集成开发环境中，除了用MQL5创建应用程序外，还可以直接从编辑器中运行Python脚本。为此，请在MetaEditor设置中指定可执行文件的路径：

© 2000-2025, MetaQuotes Ltd.




如果您的计算机上没有安装Python，请点击“安装”并下载安装文件。

您可以在MetaEdtior中创建Python脚本，或者将其上传到程序端的数据文件夹中，并使用F7 (Compile)键立即运行。这将打开MetaTrader 5程序端，脚本在当前图表上运行。来自Python控制板(stdout, stderr)的消息将显示在错误部分。

## MetaTrader 5中的模型操作

MQL5语言允许您直接在MetaTrader 5程序端中运行ONNX模型。分为三个步骤：

1. 在第三方平台（例如 Python）中训练模型。
2. 将模型转换为ONNX。
3. 使用ONNX函数将ONNX模型加入到EA交易中，并在MetaTrader 5程序端中运行。

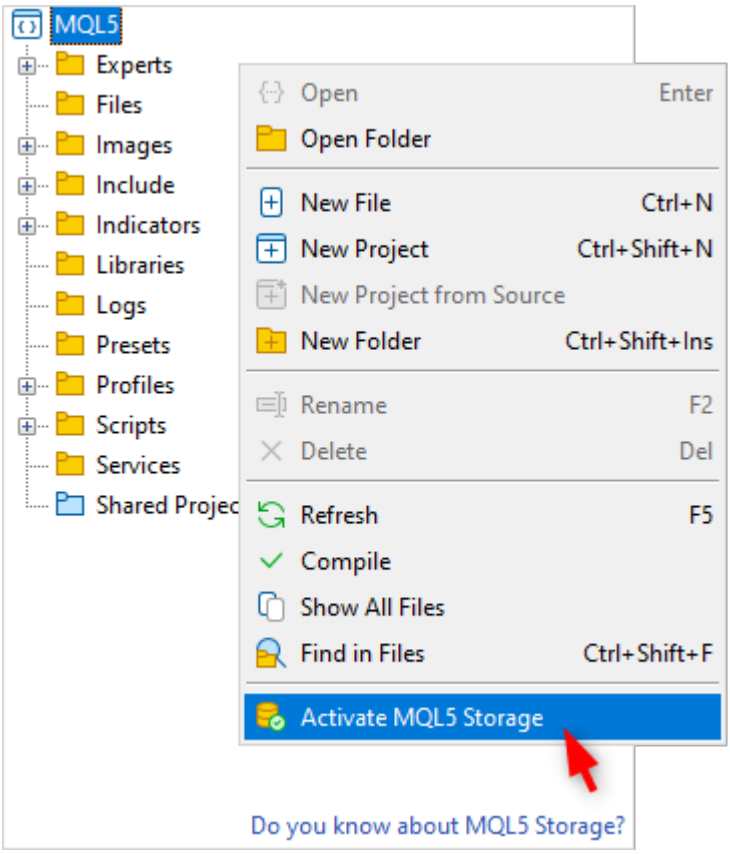
MQL5中的Python集成允许运行python脚本并在MetaEditor中保存ONNX模型或直接在MetaTrader 5图表上运行。您可以在程序端中根据需要随时使用预先编写的Python脚本来训练模型。该程序库包含用于获取价格数据的现有函数，这些数据可以输入到ONNX模型中：

- copy_rates_from – 获取从指定日期开始的柱形图
- copy_rates_from_pos – 获取从指定索引开始的柱形图
- copy_ticks_range – 获取指定日期范围的柱形图
- copy_ticks_from – 获取从指定日期开始的报价
- copy_ticks_range – 获取指定日期范围的报价

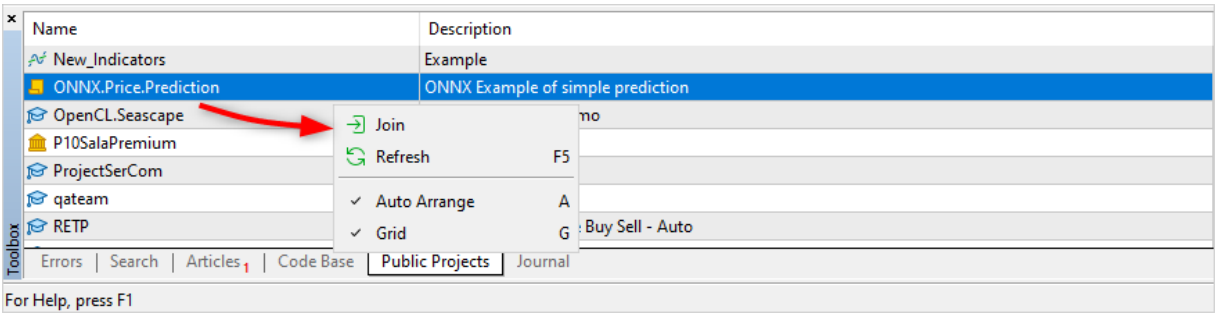
## 模型示例

© 2000-2025, MetaQuotes Ltd.

完整的ONNX模型示例可在公共项目中获得。您应该首先在导航器中通过在MetaEditor设置中指定您的MQL5登录名（区分大小写）激活MQL5存储。

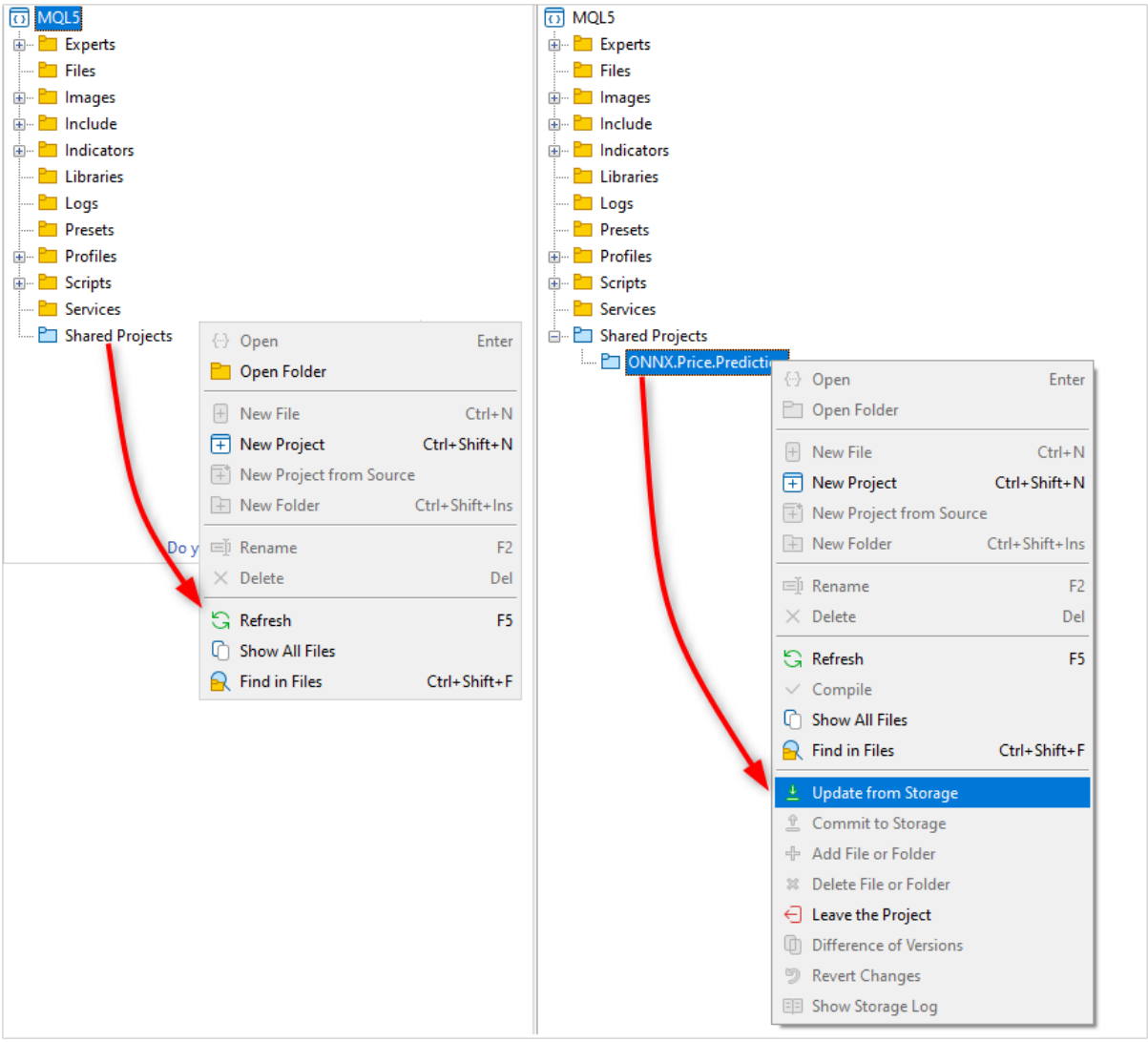



激活后，找到ONNX.Price.Prediction项目并通过快捷菜单命令加入。

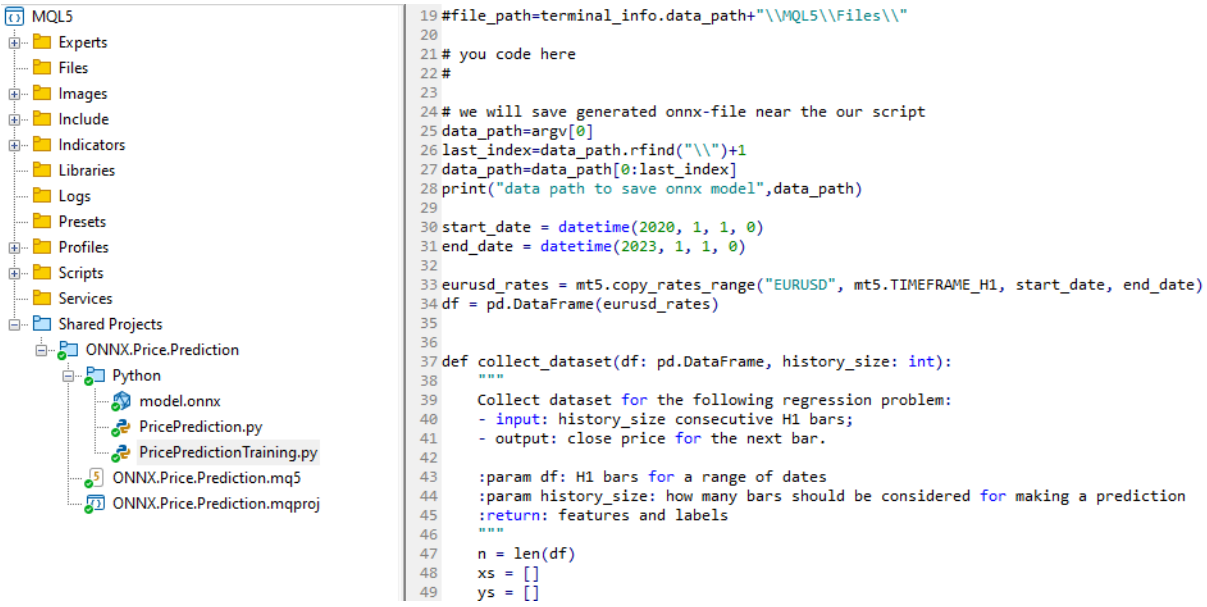



接下来，从MQL5存储更新项目。

© 2000-2025, MetaQuotes Ltd.




该项目包含一个ONNX模型、两个python脚本、一个用于项目操作的MQL5脚本和一个MQL5项目文件（ONNX.Price.Prediction.mqproj）。



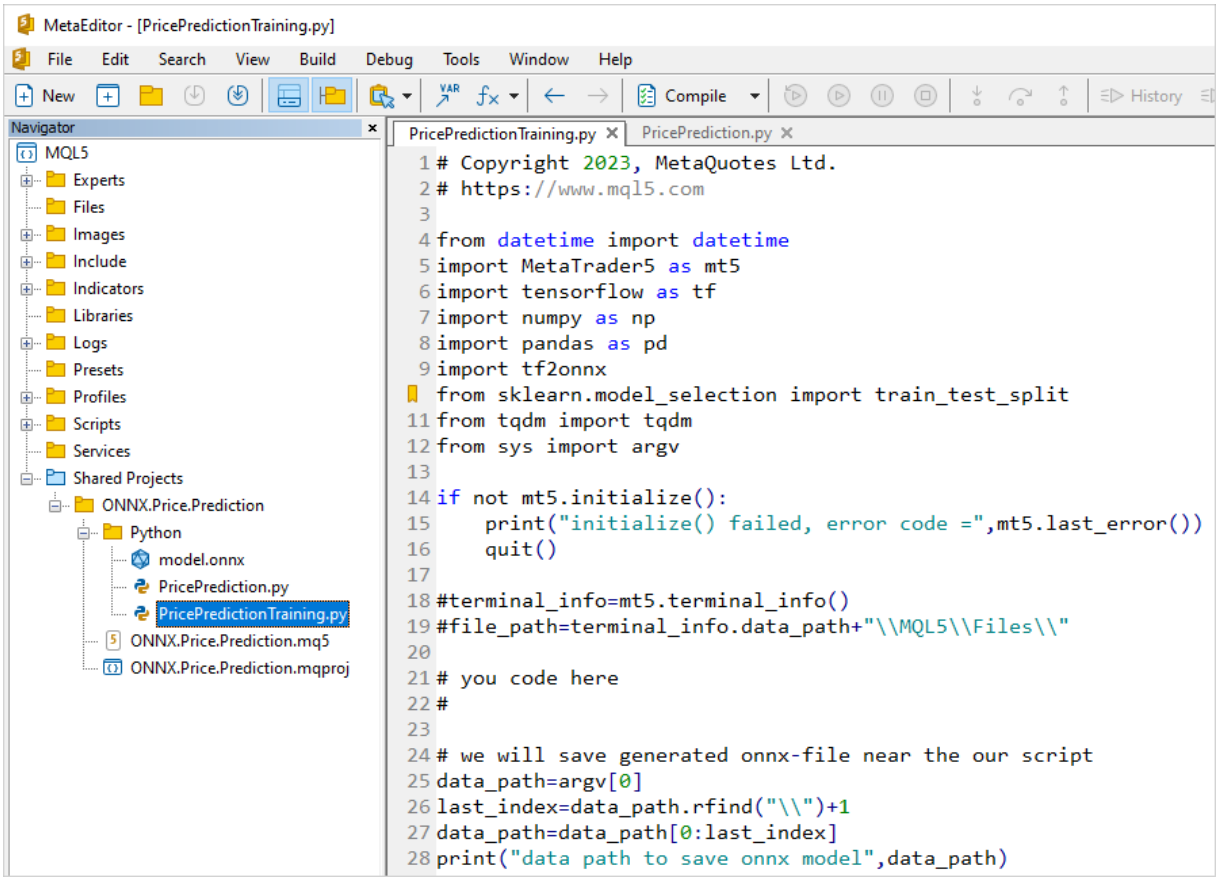

```
#file_path=terminal_info.data_path+"\\MQL5\\Files\\"

# you code here
#

# we will save generated onnx-file near the our script
data_path=argv[0]
last_index=data_path.rfind("\\")+1
data_path=data_path[0:last_index]
print("data path to save onnx model",data_path)

start_date = datetime(2020, 1, 1, 0)
end_date = datetime(2023, 1, 1, 0)

eurusd_rates = mt5.copy_rates_range("EURUSD", mt5.TIMEFRAME_H1, start_date, end_date)
df = pd.DataFrame(eurusd_rates)


def collect_dataset(df: pd.DataFrame, history_size: int):
    """
    Collect dataset for the following regression problem:
    - input: history_size consecutive H1 bars;
    - output: close price for the next bar.

    :param df: H1 bars for a range of dates
    :param history_size: how many bars should be considered for making a prediction
    :return: features and labels
    """
    n = len(df)
    xs = []
    ys = []
```

© 2000-2025, MetaQuotes Ltd.

您可以使用项目中包含的PricePredictionTraining.py脚本自行创建ONNX模型。为此，您应该首先从命令行安装所需的模块。

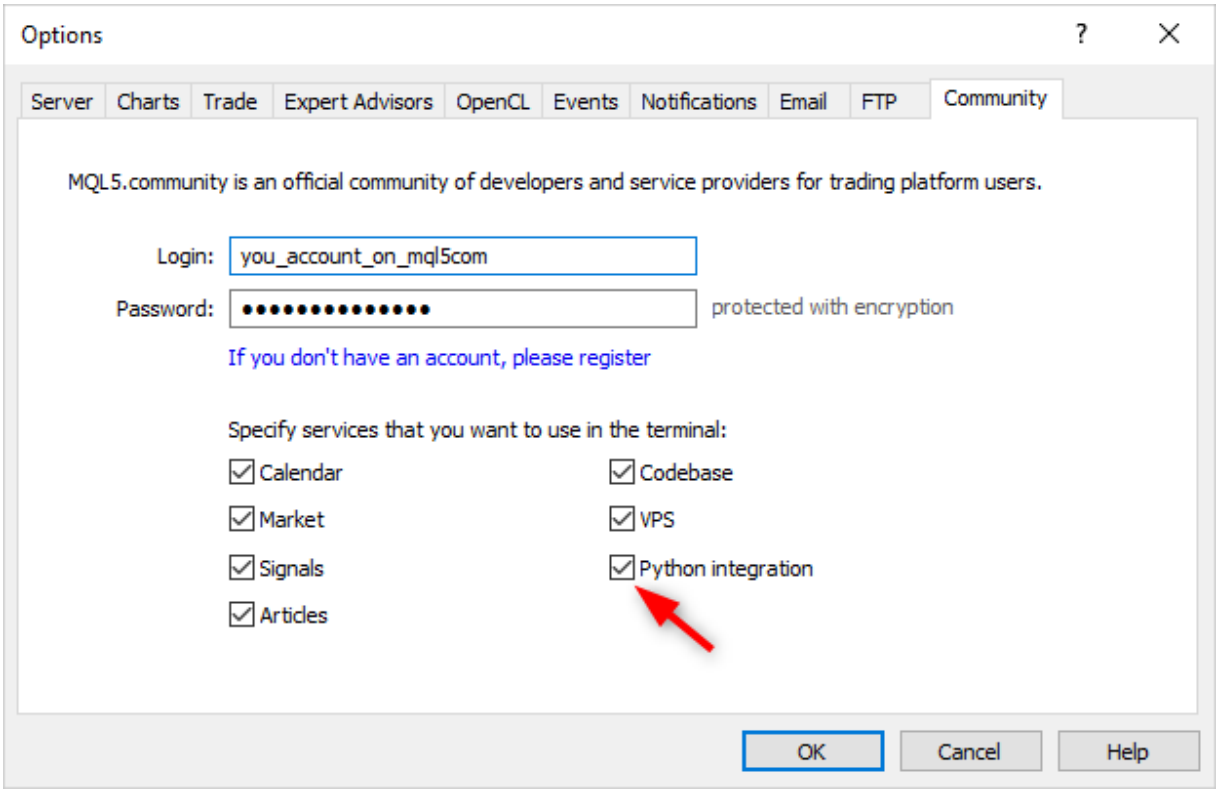
```
python.exe -m pip install --upgrade pip
python -m pip install --upgrade tensorflow
python -m pip install --upgrade pandas
python -m pip install --upgrade scikit-learn
python -m pip install --upgrade matplotlib
python -m pip install --upgrade tqdm
python -m pip install --upgrade metatrader5
python -m pip install --upgrade onnx==1.12
python -m pip install --upgrade tf2onnx
python -m pip install --upgrade numpy
python -m pip install onnxruntime
```

安装模块后，在MetaEditor中打开PricePredictionTraining.py脚本，并使用编译按键或F7键运行它。

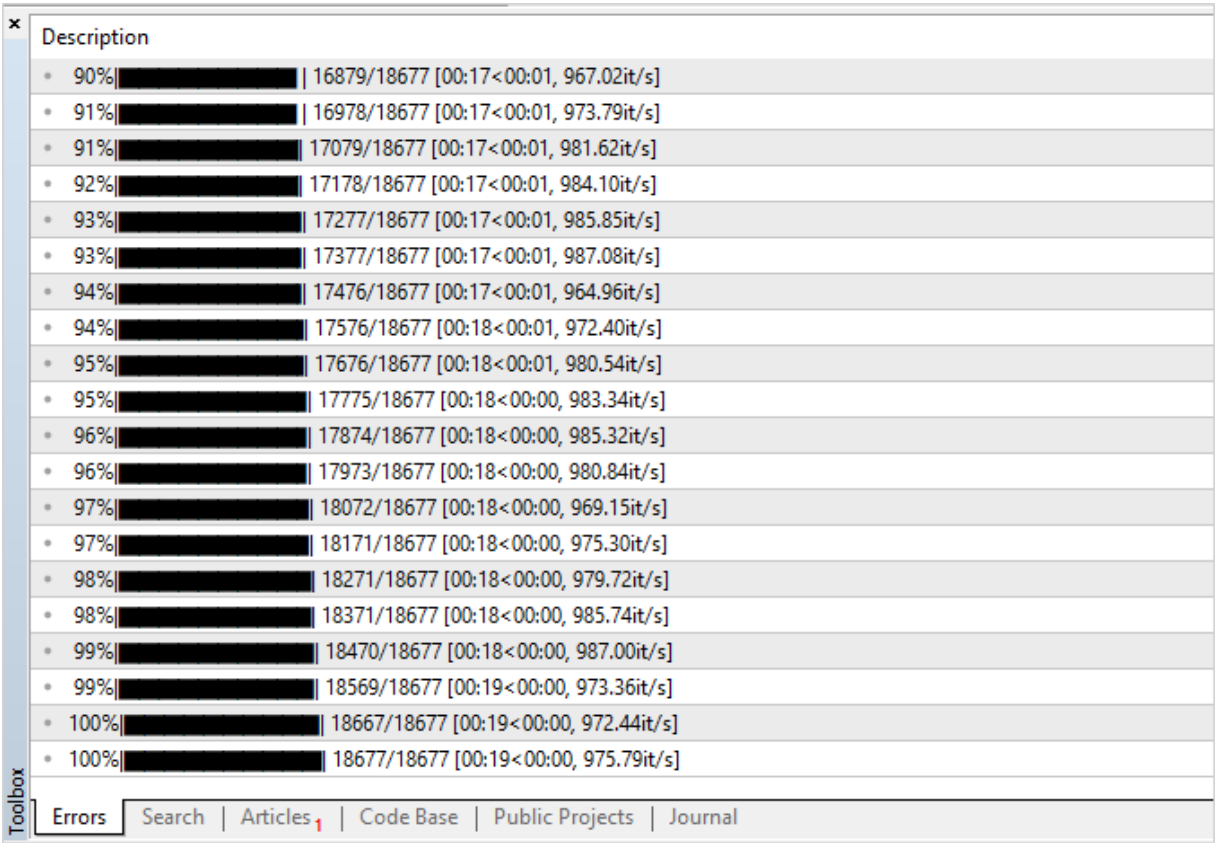



在运行Python脚本之前，请确保MetaTrader 5程序端已连接到具有EURUSD交易品种可用的服务器。例如，连接到MetaQuotes-Demo服务器，并在程序端设置中选中“与Python集成”。

© 2000-2025, MetaQuotes Ltd.



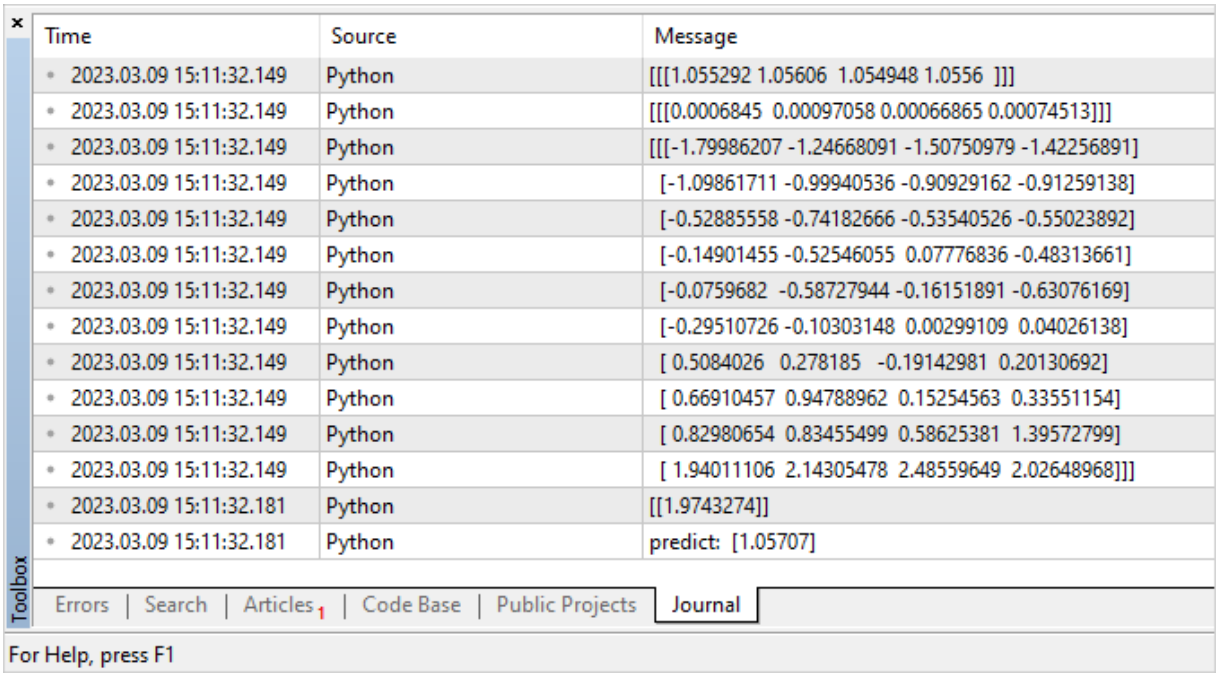

在训练网络时，MetaEditor将打印来自Python脚本的消息，直到训练完成。




© 2000-2025, MetaQuotes Ltd.

当结果为100%时，ONNX模型准备就绪，它被保存到位于<terminal data directory>\MQL5\Shared Projects\ONNX.Price.Prediction\Python的项目文件夹中。

您可以通过运行第二个脚本PricePrediction.py，按 **F7**来检查生成的模型。



| Time | Source | Message |
|---|---|---|
| 2023.03.09 15:11:32.149 | Python | [[[1.055292 1.05606 1.054948 1.0556 ]]] |
| 2023.03.09 15:11:32.149 | Python | [[[0.0006845 0.00097058 0.00066865 0.00074513]]] |
| 2023.03.09 15:11:32.149 | Python | [[[-1.79986207 -1.24668091 -1.50750979 -1.42256891] |
| 2023.03.09 15:11:32.149 | Python | [-1.09861711 -0.99940536 -0.90929162 -0.91259138] |
| 2023.03.09 15:11:32.149 | Python | [-0.52885558 -0.74182666 -0.53540526 -0.55023892] |
| 2023.03.09 15:11:32.149 | Python | [-0.14901455 -0.52546055 0.07776836 -0.48313661] |
| 2023.03.09 15:11:32.149 | Python | [-0.0759682 -0.58727944 -0.16151891 -0.63076169] |
| 2023.03.09 15:11:32.149 | Python | [-0.29510726 -0.10303148 0.00299109 0.04026138] |
| 2023.03.09 15:11:32.149 | Python | [ 0.5084026 0.278185 -0.19142981 0.20130692] |
| 2023.03.09 15:11:32.149 | Python | [ 0.66910457 0.94788962 0.15254563 0.33551154] |
| 2023.03.09 15:11:32.149 | Python | [ 0.82980654 0.83455499 0.58625381 1.39572799] |
| 2023.03.09 15:11:32.149 | Python | [ 1.94011106 2.14305478 2.48559649 2.02648968]]] |
| 2023.03.09 15:11:32.181 | Python | [[1.9743274]] |
| 2023.03.09 15:11:32.181 | Python | predict: [1.05707] |



© 2000-2025, MetaQuotes Ltd.

# 运行模型

要在MQL5中运行ONNX模型，请完成以下3个步骤：

1. 使用OnnxCreate函数从*.onnx文件中加载模型或使用OnnxCreateFromBuffer函数从数组中加载模型。
2. 使用OnnxSetInputShape和OnnxSetOutputShape函数指定输入和输出数据形状。
3. 使用OnnxRun函数运行模型，并将相关的输入和输出参数传递到这里。
4. 当需要时，您可以使用OnnxRelease函数终止模型操作。

在创建ONNX模型时，您应该考虑现有的范围和限制，请参阅
https://github.com/microsoft/onnxruntime/blob/rel-1.14.0/docs/OperatorKernels.md
这类限制的一些示例如下：

| Operation | 支持的数据类型 |
|---|---|
| ReduceSum | tensor(double), tensor(float), tensor(int32), tensor(int64) |
| Mul | tensor(bfloat16), tensor(double), tensor(float), tensor(float16), tensor(int32), tensor(int64), tensor(uint32), tensor(uint64) |

下面是来自公共项目ONNX.Price.Prediction的MQL5代码示例。

```
const long   ExtOutputShape[] = {1,1};     // 模型输出形状
const long   ExtInputShape [] = {1,10,4}; // 模型输入形状
#resource "Python/model.onnx" as uchar ExtModel[]// 作为资源的模型
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   matrix rates;
//--- 获取10个柱形图
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,2,10))
      return(-1);
//--- 输入一组OHLC向量
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- 填充归一化矩阵
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- 标准化输入数据
```

© 2000-2025, MetaQuotes Ltd.

```
   x_norm-=mm;
   x_norm/=ms;
//--- 创建模型
   long handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- 指定输入数据形状
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 指定输出数据形状
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("OnnxSetOutputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 将标准化输入数据转换为浮点类型
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- 在这里获取模型的输出数据，即价格预测
   vectorf y_norm(1);
//--- 运行模型
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 将模型的输出值打印到日志中
   Print(y_norm);
//--- 进行反向转换，以获得预测价格
   double y_pred=y_norm[0]*s[3]+m[3];
   Print("price predicted:",y_pred);
//--- 完成操作
   OnnxRelease(handle);
   return(0);
  }
```

Script run example:

```
ONNX: Creating and using per session threadpools since use_per_session_threads_ is tru
ONNX: Dynamic block base set to 0
ONNX: Initializing session.
ONNX: Adding default CPU execution provider.
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Total shared scalar initializer count: 0
```

© 2000-2025, MetaQuotes Ltd.

```
ONNX: Total fused reshape node count: 0
ONNX: Use DeviceBasedPartition as default
ONNX: Saving initialized tensors.
ONNX: Done saving initialized tensors
ONNX: Session successfully initialized.
[0.28188983]
predicted 1.0559258806393044
```

The MetaTrader 5 terminal has selected the optimal executor for calculations — ONNX Runtime Execution Provider. In this example, the model was executed on the CPU.

Let's modify the script to calculate the percentage of successful Close price predictions made based on the values of the preceding 10 bars.

```
#resource "Python/model.onnx" as uchar ExtModel[]// 作为资源的模型

#define TESTS 10000  // number of test datasets
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart()
  {
//--- 创建模型
   long session_handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
   if(session_handle==INVALID_HANDLE)
     {
      Print("Cannot create model. Error ",GetLastError());
      return(-1);
     }

//--- 由于没有为模型定义输入张量大小，请明确指定
//--- 第一个指数是批量大小，第二个指数是系列大小，第三个指数是系列数（OHLC）
   const long input_shape[]={1,10,4};
   if(!OnnxSetInputShape(session_handle,0,input_shape))
     {
      Print("OnnxSetInputShape error ",GetLastError());
      return(-2);
     }

//--- 由于没有为模型定义输出张量大小，请明确指定
//--- 第一个指数是批量大小，必须匹配输入张量的批量大小
//--- 第二个指数是预测价格的数量（这里只预测Close）
   const long output_shape[]={1,1};
   if(!OnnxSetOutputShape(session_handle,0,output_shape))
     {
      Print("OnnxSetOutputShape error ",GetLastError());
      return(-3);
     }
//--- 运行测试
   vector closes(TESTS);      // 存储验证价格的向量
```

© 2000-2025, MetaQuotes Ltd.

```
   vector predicts(TESTS);     // 存储所得预测的向量
   vector prev_closes(TESTS); // 存储之前价格的向量

   matrix rates;               // 获得OHLC系列的矩阵
   matrix splitted[2];         // 将系列分为测试和验证的两个子矩阵
   ulong  parts[]={10,1};      // 划分子矩阵的大小

//--- 从前一柱形图开始
   for(int i=1; i<=TESTS; i++)
     {
      //--- 获取11个柱形图
      rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,i,11);
      //--- 将矩阵分为测试和验证
      rates.Vsplit(parts,splitted);
      //--- 从验证矩阵得到收盘价
      closes[i-1]=splitted[1][3][0];
      //--- 测试系列中的最后收盘价
      prev_closes[i-1]=splitted[0][3][9];

      //--- 提交10个柱形图测试矩阵进行测试
      predicts[i-1]=PricePredictionTest(session_handle,splitted[0]);
      //--- 运行时间错误
      if(predicts[i-1]<=0)
        {
         OnnxRelease(session_handle);
         return(-4);
        }
     }
//--- 完成操作
   OnnxRelease(session_handle);
//--- 评估价格移动预测是否正确
   int    right_directions=0;
   vector delta_predicts=prev_closes-predicts;
   vector delta_actuals=prev_closes-closes;

   for(int i=0; i<TESTS; i++)
      if((delta_predicts[i]>0 && delta_actuals[i]>0) || (delta_predicts[i]<0 && delta_
         right_directions++;
   PrintFormat("right direction predictions = %.2f%%",(right_directions*100.0)/double
//---
   return(0);
  }
//+------------------------------------------------------------------+
//|  准备数据并运行模型                                              |
//+------------------------------------------------------------------+
double PricePredictionTest(const long session_handle,matrix& rates)
  {
   static matrixf input_data(10,4); // 用于转换输入的矩阵
   static vectorf output_data(1);   // 接收结果的向量
```

© 2000-2025, MetaQuotes Ltd.

```
   static matrix mm(10,4);           // 水平向量矩阵Mean
   static matrix ms(10,4);           // 水平向量矩阵Std

//--- 一组OHLC垂直向量必须输入到模型中
   matrix x_norm=rates.Transpose();
//--- 标准价格
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
   x_norm-=mm;
   x_norm/=ms;

//--- 运行模型
   input_data.Assign(x_norm);
   if(!OnnxRun(session_handle,ONNX_DEBUG_LOGS,input_data,output_data))
     {
      Print("OnnxRun error ",GetLastError());
      return(0);
     }
//--- 将输出值中的价格非标准化
   double y_pred=output_data[0]*s[3]+m[3];

   return(y_pred);
  }
```

运行脚本：预测准确率约为51%

```
ONNX: Creating and using per session threadpools since use_per_session_threads_ is tru
ONNX: Dynamic block base set to 0
ONNX: Initializing session.
ONNX: Adding default CPU execution provider.
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Use DeviceBasedPartition as default
ONNX: Saving initialized tensors.
ONNX: Done saving initialized tensors
ONNX: Session successfully initialized.
right direction predictions = 51.34 %
```

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# 策略测试中的模型验证

为金融市场操作创建的模型可以在 MetaTrader 5程序端策略测试部分进行验证。这是最快、最方便的选项，无需额外模拟市场环境和交易条件。

若要测试模型，需要根据公共项目ONNX.Price.Prediction的代码创建一个EA交易程序。这将需要进行一些编辑。

将模型创建移至OnInit函数。onnx会话将在OnDeinit中关闭。将主要模型操作块定位到OnTick处理程序。

此外，添加获取前两个柱形图的收盘价，需要与实际收盘价和预测收盘价进行比较。

ExpertAdvisor代码小且易于阅读。

```
const long   ExtInputShape [] = {1,10,4}; // 模型输入形状
const long   ExtOutputShape[] = {1,1};     // 模型输出形状
#resource "Python/model.onnx" as uchar ExtModel[];// 作为资源的模型

long handle;          // 模型句柄
ulong predictions=0; // 预测计数器
ulong confirmed=0;   // 成功预测计数器
//+------------------------------------------------------------------+
//| EA交易初始化函数                                                 |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- 基本检查
   if(_Symbol!="EURUSD")
     {
      Print("Symbol must be EURUSD, testing aborted");
      return(-1);
     }
   if(_Period!=PERIOD_H1)
     {
      Print("Timeframe must be H1, testing aborted");
      return(-1);
     }
//--- 创建模型
   handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- 指定输入数据形状
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 指定输出数据形状
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
```

© 2000-2025, MetaQuotes Ltd.

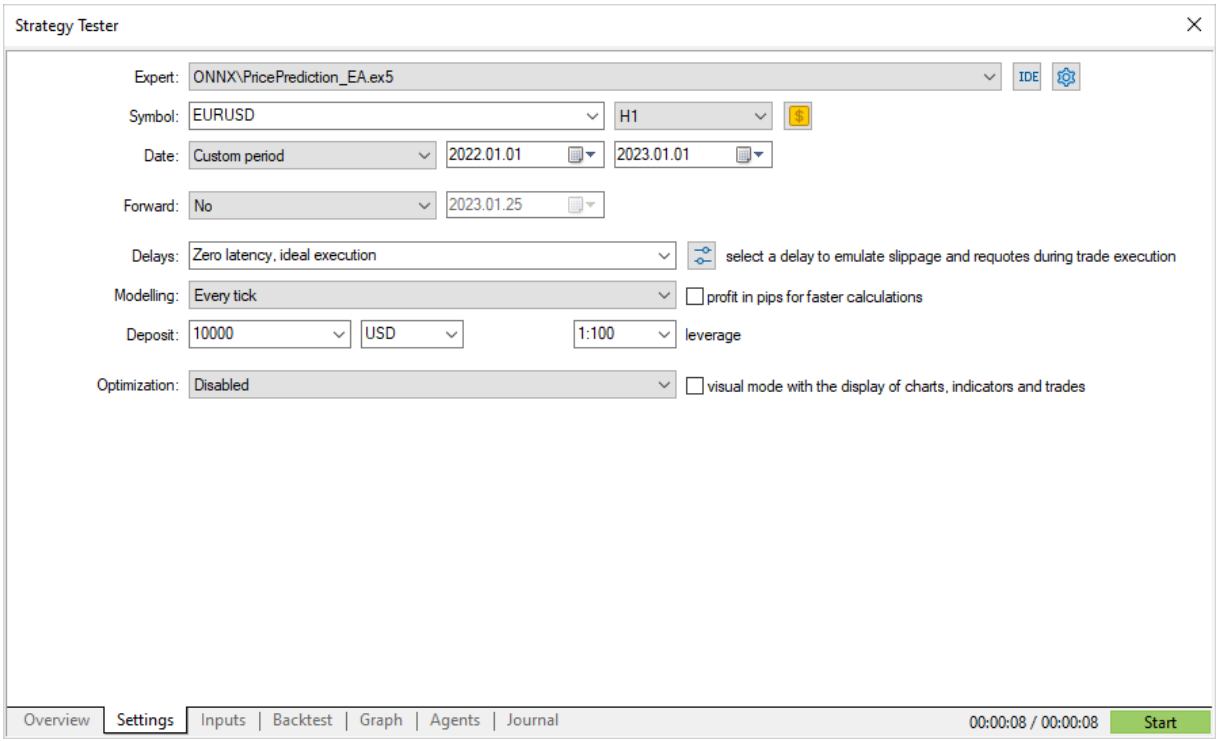
```
      {
       Print("OnnxSetOutputShape failed, error ",GetLastError());
       OnnxRelease(handle);
       return(-1);
      }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| EA交易去初始化函数                                               |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- 完成模型操作
   OnnxRelease(handle);
//--- 计算和输出预测统计
   PrintFormat("Successfull predictions = %.2f %%",confirmed*100./double(predictions));
  }
//+------------------------------------------------------------------+
//| EA报价函数                                                       |
//+------------------------------------------------------------------+
void OnTick()
  {
   static datetime open_time=0;
   static double predict;
//--- 检查当前柱形图开盘时间
   datetime time=iTime(_Symbol,_Period,0);
   if(time==0)
     {
      PrintFormat("Failed to get Time(0), error %d", GetLastError());
      return;
     }
//--- 如果开盘时间没有改变，则退出直至下一次OnTick调用
   if(time==open_time)
      return;
//--- 获取最后两个已完成柱形图的收盘价
   double close[];
   int recieved=CopyClose(_Symbol,_Period,1,2,close);
   if(recieved!=2)
     {
      PrintFormat("CopyClose(2 bars) failed, error %d",GetLastError());
      return;
     }
   double delta_predict=predict-close[0]; // predicted price change
   double delta_actual=close[1]-close[0]; // actual price change
   if((delta_predict>0 && delta_actual>0) || (delta_predict<0 && delta_actual<0))
      confirmed++;

//--- 计算新柱形图上的收盘价，以验证下一个柱形图上的价格
```

© 2000-2025, MetaQuotes Ltd.

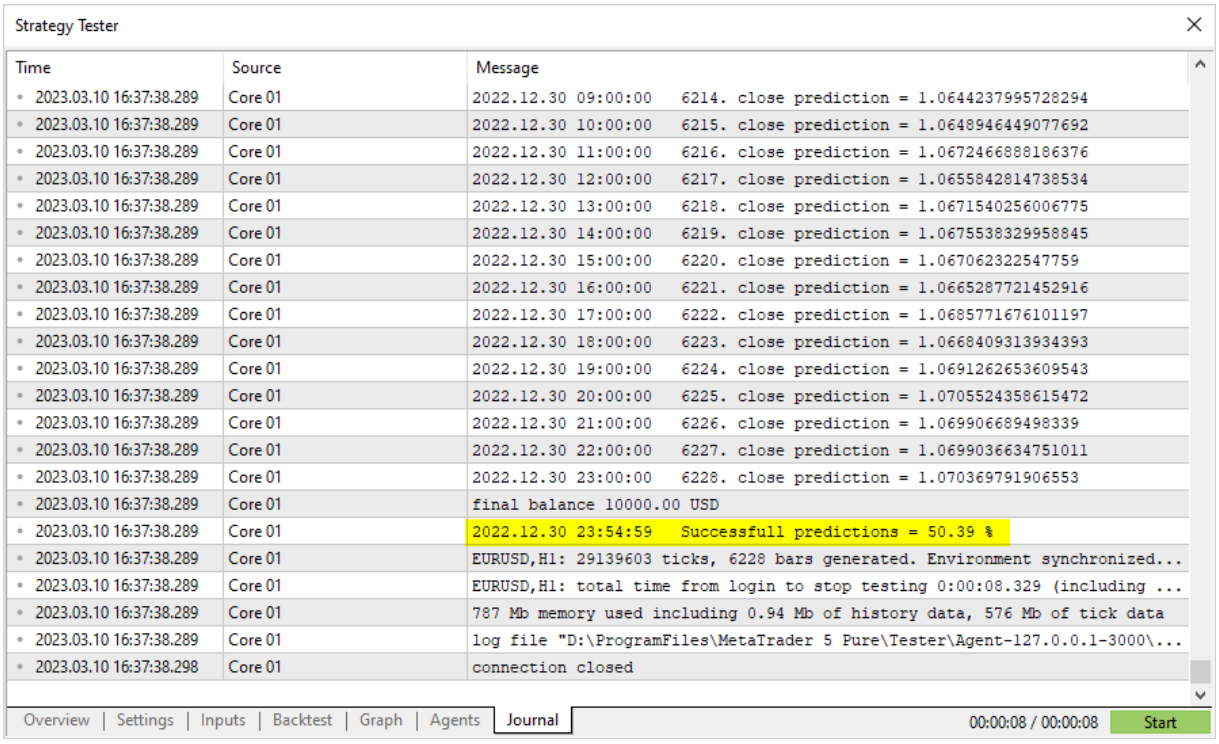
```
   matrix rates;
//--- 获取10个柱形图
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,1,10))
      return;
//--- 输入一组OHLC向量
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- 填充归一化矩阵
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- 标准化输入数据
   x_norm-=mm;
   x_norm/=ms;
//--- 将标准化输入数据转换为浮点类型
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- 在这里获取模型的输出数据，即价格预测
   vectorf y_norm(1);
//--- 运行模型
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
     }
//--- 进行反向转换，以获得预测价格并在新柱形图上对其验证
   predict=y_norm[0]*s[3]+m[3];
   predictions++;  // 增加预测计数器
   Print(predictions,". close prediction = ",predict);
//--- 保存柱形图开盘时间，以检查下一个报价
   open_time=time;
  }
```

编译EA交易，并在2022年期间运行测试。使用H1时间周期指定EURUSD，这是训练模型的数据。可以忽略报价建模模式，因为代码检查的是新柱形图的出现。

© 2000-2025, MetaQuotes Ltd.




运行并在测试日志中查看结果。它表明略高于50%的预测在2022年是正确的。




如果初步模型测试产生令人满意的结果，您可以开始基于此模型编写完整的交易策略。

© 2000-2025, MetaQuotes Ltd.

# OnnxCreate

创建ONNX会话，从*.onnx文件中加载一个模型。

```
long  OnnxCreate(
   string  filename,  // 文件路径
   uint    flags      //创建模型的标识
   );
```

参数

*filename*

[in] Path to the *.onnx file of the model 相对于\MQL5\Files\文件夹的到模型*.onnx文件的路径。

*flags*

[in] ENUM_ONNX_FLAGS中的标识，描述模型创建模式：ONNX_COMMON_FOLDER和ONNX_DEBUG_LOGS。

返回值

创建会话的句柄，或者出现错误时的INVALID_HANDLE。要获取错误代码，请调用GetLastError函数。

注意

如果在磁盘上找不到指定的文件，系统将尝试重新打开文件，并在文件名后附加“.onnx”扩展名。

© 2000-2025, MetaQuotes Ltd.

# OnnxCreateFromBuffer

创建ONNX会话，从数据数组加载模型。

```
long  OnnxCreateFromBuffer(
   const uchar&  buffer[],   // 数组参考
   ulong         flags       // 模型创建标识
   );
```

参数

*buffer*

[in]使用ONNX模型数据的数组。

*flags*

[in] ENUM_ONNX_FLAGS中的标识，描述模型创建模式：ONNX_COMMON_FOLDER和ONNX_DEBUG_LOGS。

返回值

创建会话的句柄，或者出现错误时的INVALID_HANDLE。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxRelease

关闭ONNX会话。

```
bool  OnnxRelease(
   long   onnx_handle  //ONNX会话句柄
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

返回值

如果成功返回true；否则返回false。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxRun

运行ONNX模型。

```
bool  OnnxRun(
   long     onnx_handle,  // ONNX会话句柄
   ulong    flags,        // 描述运行模式的标识
   ...                    // 模型的输入和输出
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*flags*

[in] ENUM_ONNX_FLAGS中的标识，描述运行模式：ONNX_DEBUG_LOGS和ONNX_NO_CONVERSION。

*...*

[in][out] 模型输入和输出。

成功返回true，否则返回false。要获取错误代码，请调用GetLastError函数。

**ENUM_ONNX_FLAGS**

| ID | 描述 |
|---|---|
| ONNX_DEBUG_LOGS | 输出调试日志 |
| ONNX_NO_CONVERSION | 禁用自动转换，按原样使用用户数据 |
| ONNX_COMMON_FOLDER | 加载Common\Files文件夹的模型文件；该值等于FILE_COMMON标识 |

例如：

```
const long                              ExtOutputShape[] = {1,1};     // 模型输出形状
const long                              ExtInputShape [] = {1,10,4}; // 模型输入形状
#resource "Python/model.onnx" as uchar ExtModel[]                    // 作为资源的模型
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   matrix rates;
//--- 获取10个柱形图
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,2,10))
      return(-1);
//--- 输入一组OHLC向量
```

© 2000-2025, MetaQuotes Ltd.

```
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- 填充归一化矩阵
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- 标准化输入数据
   x_norm-=mm;
   x_norm/=ms;
//--- 创建模型
   long handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- 指定输入数据形状
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 指定输出数据形状
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("OnnxSetOutputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 将标准化输入数据转换为浮点类型
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- 在这里获取模型的输出数据，即价格预测
   vectorf y_norm(1);
//--- 运行模型
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 将模型的输出值打印到日志中
   Print(y_norm);
//--- 进行反向转换，以获得预测价格
   double y_pred=y_norm[0]*s[3]+m[3];
   Print("price predicted:",y_pred);
//--- 已完成操作
   OnnxRelease(handle);
```

© 2000-2025, MetaQuotes Ltd.

```
   return(0);
  };
```

另见

OnnxSetInputShape, OnnxSetOutputShape

© 2000-2025, MetaQuotes Ltd.

# OnnxGetInputCount

获取ONNX模型中的输入数。

```
long  OnnxGetInputCount(
   long   onnx_handle  //ONNX会话句柄
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

返回值

如果成功，返回输入参数的数量；否则返回-1。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxGetOutputCount

Get the number of outputs in an ONNX model.

```
long  OnnxGetOutputCount(
   long   onnx_handle  //ONNX会话句柄
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

返回值

如果成功，返回输出参数的数量；否则返回-1。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxGetInputName

通过索引获取模型输入名称

```
string  OnnxGetInputName(
   long   onnx_handle,  // ONNX会话句柄
   long   index         // 参数索引
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*index*

[in] 输入参数的索引，从0开始。

返回值

如果成功，返回输入参数的名称；否则返回NULL。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxGetOutputName

通过索引获取模型输出名称。

```
string  OnnxGetOutputName(
   long   onnx_handle,  // ONNX会话句柄
   long   index         // 参数索引
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*index*

[in] 输出参数的索引，从0开始。

返回值

如果成功，返回输出参数的名称；否则返回NULL。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxGetInputTypeInfo

从模型获取输入类型的描述

```
bool  OnnxGetInputTypeInfo(
   long           onnx_handle,  // ONNX会话句柄
   long           index,        // 参数索引
   OnnxTypeInfo&  typeinfo      // 参数类型描述
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*index*

[in] 输入参数的索引，从0开始。

*typeinfo*

[out] 描述输入参数类型的OnnxTypeInfo结构。

返回值

如果成功返回true；否则返回false。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxGetOutputTypeInfo

从模型获取输出类型的描述。

```
bool  OnnxGetOutputTypeInfo(
   long           onnx_handle,  // ONNX会话句柄
   long           index,        // 参数索引
   OnnxTypeInfo&  typeinfo      // 参数类型描述
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*index*

[in] 输出参数的索引，从0开始。

*typeinfo*

[out] 描述输出参数类型的OnnxTypeInfo结构。

返回值

如果成功返回true；否则返回false。要获取错误代码，请调用GetLastError函数。

© 2000-2025, MetaQuotes Ltd.

# OnnxSetInputShape

通过索引设置模型输入数据的形状。

```
bool  OnnxSetInputShape(
   long           onnx_handle,   // ONNX会话句柄
   long           input_index,   // 输入参数索引
   const ulong&   shape[]        // 描述输入数据形状的数组
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*input_index*

[in] 输入参数的索引，从0开始。

*shape*

[in] 描述模型输入数据形状的数组。

返回值

如果成功，返回输入参数的名称；否则返回NULL。要获取错误代码，请调用GetLastError函数。

例如：

```
//---- 描述模型输入和输出数据的形状
   const long  ExtOutputShape[] = {1,1};
   const long  ExtInputShape [] = {1,10,4};
//--- 创建模型
   long handle=OnnxCreateFromBuffer(model,ONNX_DEBUG_LOGS);
//--- 指定输入数据形状
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("failed, OnnxSetInputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 指定输出数据形状
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("failed, OnnxSetOutputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
```

另见

OnnxSetOutputShape

© 2000-2025, MetaQuotes Ltd.

# OnnxSetOutputShape

通过索引设置模型输出数据的形状。

```
bool  OnnxSetOutputShape(
   long           onnx_handle,    // ONNX会话句柄
   long           output_index,   // 输出参数索引
   const ulong&   shape[]         // 描述输出数据形状的数组
   );
```

参数

*onnx_handle*

[in] 通过OnnxCreate或OnnxCreateFromBuffer创建的ONNX会话对象句柄。

*output_index*

[in] 输出参数的索引，从0开始。

*shape*

[in] 描述模型输出数据形状的数组。

返回值

如果成功，返回输入参数的名称；否则返回NULL。要获取错误代码，请调用GetLastError函数。

例如：

```
//---- 描述模型输入和输出数据的形状
   const long  ExtOutputShape[] = {1,1};
   const long  ExtInputShape [] = {1,10,4};
//--- 创建模型
   long handle=OnnxCreateFromBuffer(model,ONNX_DEBUG_LOGS);
//--- 指定输入数据形状
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("failed, OnnxSetInputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- 指定输出数据形状
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("failed, OnnxSetOutputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
```

另见

OnnxSetInputShape

© 2000-2025, MetaQuotes Ltd.

# 数据结构

以下数据结构用于ONNX模型的操作：

## OnnxTypeInfo

该结构描述了ONNX模型的输入或输出参数的类型

```
struct OnnxTypeInfo
 {
  ENUM_ONNX_TYPE         type;           // 参数类型
  OnnxTensorTypeInfo     tensor;         // 张量描述
  OnnxMapTypeInfo        map;            // 映射描述
  OnnxSequenceTypeInfo   sequence;       // 序列描述
 };
```

只有张量(ONNX_TYPE_TENSOR)可以作为输入使用。在这种情况下，只有OnnxTypeInfo::tensor字段被填充值，而其他字段（映射和序列）没有定义。

三种OnnxTypeInfo类型（ONNX_TYPE_TENSOR、ONNX_TYPE_MAP或ONNX_TYPE_SEQUENCE）中只有一种可以作为输入使用。相对应的子结构（OnnxTypeInfo::tensor、OnnxTypeInfo::map或OnnxTypeInfo::sequence）根据类型来填充。

## OnnxTensorTypeInfo

该结构描述了ONNX模型的输入或输出参数中的张量

```
struct OnnxTensorTypeInfo
 {
  const ENUM_ONNX_DATA_TYPE  data_type;      // 张量中的数据类型
  const long                 dimensions[];   // 张量中的元素数
 };
```

## OnnxMapTypeInfo

该结构描述了ONNX模型的输出参数中获得的映射

```
struct OnnxMapTypeInfo
 {
  const ENUM_ONNX_DATA_TYPE  key_type;       //密钥类型
  const OnnxTypeInfo&        value_type;     // 值类型
 };
```

## OnnxSequenceTypeInfo

该结构描述了ONNX模型的输出参数中获得的序列

```
struct OnnxSequenceTypeInfo
 {
```

© 2000-2025, MetaQuotes Ltd.

```
  const OnnxTypeInfo&        value_type;      // 序列中的数据类型
};
```

## ENUM_ONNX_TYPE

**ENUM_ONNX_TYPE**枚举描述了模型参数的类型

| ID | 描述 |
|---|---|
| ONNX_TYPE_UNKNOWN | 未知 |
| ONNX_TYPE_TENSOR | Tensor |
| ONNX_TYPE_SEQUENCE | Sequence |
| ONNX_TYPE_MAP | Map |
| ONNX_TYPE_OPAQUE | Abstract (opaque) |
| ONNX_TYPE_SPARSETENSOR | Sparse tensor |

## ENUM_ONNX_DATA_TYPE

**ENUM_ONNX_DATA_TYPE**枚举描述了所用数据的类型

| ID | 描述 |
|---|---|
| ONNX_DATA_TYPE_UNDEFINED | 未定义 |
| ONNX_DATA_TYPE_FLOAT | 浮点型 |
| ONNX_DATA_TYPE_INT8 | 8位整型 |
| ONNX_DATA_TYPE_UINT16 | 16位无符号整型 |
| ONNX_DATA_TYPE_INT16 | 16位整型 |
| ONNX_DATA_TYPE_INT32 | 32位整型 |
| ONNX_DATA_TYPE_INT64 | 64位整型 |
| ONNX_DATA_TYPE_STRING | 字符串 |
| ONNX_DATA_TYPE_BOOL | 布尔型 |
| ONNX_DATA_TYPE_FLOAT16 | 16位浮点型 |
| ONNX_DATA_TYPE_DOUBLE | 双精度 |
| ONNX_DATA_TYPE_UINT32 | 32位无符号整型 |
| ONNX_DATA_TYPE_UINT64 | 64位无符号整型 |
| ONNX_DATA_TYPE_COMPLEX64 | 64位复数 |

© 2000-2025, MetaQuotes Ltd.

| ID | 描述 |
| --- | --- |
| ONNX_DATA_TYPE_COMPLEX128 | 128位复数 |
| ONNX_DATA_TYPE_BFLOAT16 | 16位bfloat（大脑浮点） |

## ENUM_ONNX_FLAGS

**ENUM_ONNX_FLAGS**枚举描述了模型运行模式

| ID | 描述 |
| --- | --- |
| ONNX_DEBUG_LOGS | 输出调试日志 |
| ONNX_NO_CONVERSION | 禁用自动转换，按原样使用用户数据 |
| ONNX_COMMON_FOLDER | 加载Common\Files文件夹的模型文件；该值等于FILE_COMMON标识 |

## 使用ONNX模型时的数组转换

机器学习任务并不总是需要更高的计算精度。为了加快计算速度，一些模型使用较低精度的数据类型，例如Float16甚至Float8。为了允许用户将相关数据输入到模型中，MQL5提供了四个特殊函数，可将标准MQL5类型转换为特殊的FP16和FP8类型。

| 函数 | 操作 |
| --- | --- |
| ArrayToFP16 | 将float或double类型数组复制到给定格式的ushort类型的数组中 |
| ArrayToFP8 | 将float或double类型数组复制到给定格式的uchar类型的数组中 |
| ArrayFromFP16 | 将ushort类型数组复制到给定格式的float或double类型的数组中 |
| ArrayFromFP8 | 将uchar类型数组复制到给定格式的float或double类型的数组中 |

这些数组转换函数使用以下枚举中指定的特殊格式。

## ENUM_FLOAT16_FORMAT

**ENUM_FLOAT16_FORMAT**枚举描述了两种FP16类型格式。

| ID | 描述 |
| --- | --- |
| FLOAT_FP16 | 标准16位格式，也称为half |
| FLOAT_BFP16 | 特殊的brain float point（大脑浮点）格式 |

© 2000-2025, MetaQuotes Ltd.

这些格式中的每一种都有其优点和局限性。FLOAT16提供更高的精度，但需要更多的存储和计算资源。另一方面，BFLOAT16在数据处理方面提供了更高的性能和效率，但可能不太准确。

## ENUM_FLOAT8_FORMAT

**ENUM_FLOAT8_FORMAT**枚举描述了四种FP8类型格式。

FP8（8位浮点）是用于表示浮点数的数据类型之一。在FP8中，每个数字由8个数据位表示，通常分为三个部分：符号、指数和尾数。这种格式提供了准确性和存储效率之间的平衡，使其对于需要内存和计算效率的应用程序具有吸引力。

| ID | 描述 |
| --- | --- |
| FLOAT_FP8_E4M3FN | 8位浮点数，4位指数，3位尾数。通常用作系数。 |
| FLOAT_FP8_E4M3FNUZ | 8位浮点数，4位指数，3位尾数。支持NaN，不支持负零和Inf。通常用作系数。 |
| FLOAT_FP8_E5M2FN | 8位浮点数，5位指数，2位尾数。支持NaN和Inf。通常用于渐变。 |
| FLOAT_FP8_E5M2FNUZ | 8位浮点数，5位指数，2位尾数。支持NaN，不支持负零和Inf。也用于渐变。 |

FP8的主要优势之一是其处理大型数据集的效率。通过采用紧凑型数字表示法，FP8减少了内存需求并加速了计算。这对于经常处理大型数据集的机器学习和人工智能应用程序尤其重要。

© 2000-2025, MetaQuotes Ltd.

# 标准库

这一组章节包含 MQL5 标准库的技术细节，以及它的所有关键组件的描述。

MQL5 标准库以 MQL5 语言编写，它设计用于为终端用户便利地编写程序（指标，脚本，智能交易）。程序库可供方便地访问众多 MQL5 的内部函数。

MQL5 标准库位于终端工作目录的 'Include' 文件夹。

| 章节 | 位置 |
| --- | --- |
| 数学 | Include\Math\ |
| OpenCL | Include\OpenCL\ |
| 基本类CObject | Include\ |
| 数据采集 | Include\Arrays\ |
| 通用数据集 | Include\Generic\ |
| 文件 | Include\Files\ |
| 字符串 | Include\Strings\ |
| 图形对象 | Include\Objects\ |
| 自定义图形 | Include\Canvas\ |
| 3D Graphics | Include\Canvas\ |
| 价格图表 | Include\Charts\ |
| 科学图表 | Include\Graphics\ |
| 指标 | Include\Indicators\ |
| 交易类 | Include\Trade\ |
| 策略模块 | Include\Expert\ |
| 面板和对话框 | Include\Controls\ |

© 2000-2025, MetaQuotes Ltd.

# 数学

提供多种程序库执行多个领域的数学计算：

- 统计 -用于处理概率理论各种分布的函数
- 模糊理论 -实施Mamdani和Sugeno模糊推理系统的程序库
- ALGLIB - 数据分析（集群，决策树，线性回归，神经网络），解决微分方程式，傅里叶变换，数值积分，优化问题，统计分析等等。

© 2000-2025, MetaQuotes Ltd.

# 统计

统计库提供了一种处理基本统计分布的便利方式。

统计库为每个分布提供5个函数：

1. 计算概率密度 -MathProbabilityDensityX()函数
2. 计算概率 -MathCumulativeDistributionX()函数
3. 计算分布四分位数 -MathQuantileX()函数
4. 生成指定分布的随机数 -MathRandomX()函数
5. 计算分布的理论力矩 -MathMomentsX()函数

除了计算各地随机变量值以外，该统计库还提供重载执行相同数组计算的函数。

- 统计特性
- 正态分布
- 对数正态分布
- Beta分布
- 非中心beta分布
- 伽玛分布
- 卡方分布
- 非中心卡方分布
- 指数分布
- F-分布
- 非中心F-分布
- t-分布
- 非中心t-分布
- 逻辑斯谛分布
- 柯西分布
- 均匀分布
- 威布尔分布
- 二项式分布
- 负二项式分布
- 几何分布
- 超几何分布
- 泊松分布
- 子函数

示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                     NormalDistributionExample.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- 包括计算正态分布的函数
#include <Math\Stat\Normal.mqh>
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 设置正态分布的参数
   double mu=5.0;
   double sigma=1.0;
   PrintFormat("Normal distribution with parameters mu=%G and sigma=%G, calculation ex
//--- 设置间隔
   double x1=mu-sigma;
   double x2=mu+sigma;
//--- 概率计算变量
   double cdf1,cdf2,probability;
//--- 错误代码变量
   int error_code1,error_code2;
//--- 计算分布函数值
   cdf1=MathCumulativeDistributionNormal(x1,mu,sigma,error_code1);
   cdf2=MathCumulativeDistributionNormal(x2,mu,sigma,error_code2);
//--- 检查错误代码
   if(error_code1==ERR_OK && error_code2==ERR_OK)
     {
      //--- 计算范围内随机变量概率
      probability=cdf2-cdf1;
      //--- 输出结果
      PrintFormat("1.  计算%.5f范围内随机变量的概率<x<%.5f",x1,x2);
      PrintFormat("  Answer: Probability = %5.8f",probability);
     }

//--- 对应95%的置信度，寻找随机变量x的值范围
   probability=0.95; //   设置置信概率
//--- 在区间范围设置概率
   double p1=(1.0-probability)*0.5;
   double p2=probability+(1.0-probability)*0.5;
//--- 计算区间范围
   x1=MathQuantileNormal(p1,mu,sigma,error_code1);
   x2=MathQuantileNormal(p2,mu,sigma,error_code2);
//--- 检查错误代码
   if(error_code1==ERR_OK && error_code2==ERR_OK)
     {
      //--- 输出结果
      PrintFormat("2.  如果置信区间 = %.2f，请查找随机变量的范围",probability);
      PrintFormat("  Answer: range is  %5.8f <= x <=%5.8f",x1,x2);
     }

   PrintFormat("3.  估算分布的前4个已计算的理论力矩");
//--- 生成随机数数组，计算前4力矩和比较理论值
   int data_count=1000000;  // 设置值数和准备数组
   double data[];
   ArrayResize(data,data_count);
//--- 生成随机值并将其存入数组
```

```
   for(int i=0; i<data_count; i++)
     {
      data[i]=MathRandomNormal(mu,sigma,error_code1);
     }
//--- 设置初始值指数和计算的数据量
   int start=0;
   int count=data_count;
//--- 计算生成值的前4力矩
   double mean=MathMean(data,start,count);
   double variance=MathVariance(data,start,count);
   double skewness=MathSkewness(data,start,count);
   double kurtosis=MathKurtosis(data,start,count);
//--- 理论力矩变量
   double normal_mean=0;
   double normal_variance=0;
   double normal_skewness=0;
   double normal_kurtosis=0;
//--- 显示计算的力矩值
   PrintFormat("            Mean           Variance          Skewness           Kurtos
   PrintFormat("Calculated  %.10f    %.10f      %.10f      %.10f",mean,variance,skewnes
//--- 计算力矩理论值并将其与获得的值相比较
   if(MathMomentsNormal(mu,sigma,normal_mean,normal_variance,normal_skewness,normal_ku
     {
      PrintFormat("Theoretical %.10f    %.10f      %.10f       %.10f",normal_mean,norma
      PrintFormat("Difference  %.10f    %.10f      %.10f       %.10f",mean-normal_mean,
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# 统计特性

这组函数计算数组元素的统计特性：

- 平均值，
- 方差，
- 偏度，
- 峰度，
- 中值，
- 均方根和
- 标准偏差。

| 函数 | 描述 |
| --- | --- |
| MathMean | 计算数组元素的平均值（第一力矩） |
| MathVariance | 计算数组元素的方差（第二力矩） |
| MathSkewness | 计算数组元素的偏度（第三力矩） |
| MathKurtosis | 计算数组元素的峰度（第四力矩） |
| MathMoments | 计算数组元素的前4力矩（平均值，方差，偏度，峰度） |
| MathMedian | 计算数组元素的中值 |
| MathStandardDeviation | 计算数组元素的标准偏差 |
| MathAverageDeviation | 计算数组元素的平均绝对偏差 |

© 2000-2025, MetaQuotes Ltd.

# MathMean

计算数组元素的平均值（第一力矩）。R语言的mean()模拟。

```
double  MathMean(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算平均值的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的平均值。出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

# MathVariance

计算数组元素的方差（第二力矩）。R语言的var()模拟。

```
double  MathVariance(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的方差。出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

# MathSkewness

计算数组元素的偏度（第三力矩）。模拟R(e1071程序库)的skewness()。

```
double  MathSkewness(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

Skewness of array elements. 出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

# MathKurtosis

计算数组元素的峰度（第四力矩）。模拟R(e1071 程序库)的kurtosis()。

```
double  MathKurtosis(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的峰度。出错情况下它返回NaN（非数字）。

免责声明

峰度计算是使用围绕正态分布的过量峰度来执行（过量峰度=峰度-3），例如正态分布的过量峰度为零。

如果分布峰值围绕期望值很尖锐则为正值，如果峰值很平坦为负值。

© 2000-2025, MetaQuotes Ltd.

# MathMoments

计算数组元素的前4力矩（平均值，方差，偏度，峰度）。

```
double  MathMoments(
   const double&  array[],              // 数据数组
   double&        mean,                 // 平均值（第1力矩）
   double&        variance,             // 方差（第2力矩）
   double&        skewness,             // 偏度（第3力矩）
   double&        kurtosis,             // 峰度（第4力矩）
   const int      start=0,              // 初始计算索引
   const int      count=WHOLE_ARRAY     // 元素数量
   );
```

参数

*数组*

[in] 计算的数据数组。

*平均值*

[out] 平均值变量（第1力矩）

*方差*

[out] 方差变量（第2力矩）

*偏度*

[out] 偏度变量（第3力矩）

*峰度*

[out] 峰度变量（第4力矩）

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

如果力矩成功计算返回true，否则为False。

免责声明

峰度计算是使用围绕正态分布的过量峰度来执行（过量峰度=峰度-3），例如正态分布的过量峰度为零。

如果分布峰值围绕期望值很尖锐则为正值，如果峰值很平坦为负值。

© 2000-2025, MetaQuotes Ltd.

# MathMedian

计算数组元素的中值。R语言的median()模拟。

```
double  MathMedian(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的中值。出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

# MathStandardDeviation

计算数组元素的标准偏差。R语言的sd()模拟。

```
double  MathStandardDeviation(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的标准偏差。出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

# MathAverageDeviation

计算数组元素的平均绝对偏差。R语言的aad()模拟。

```
double  MathAverageDeviation(
   const double&  array[]                // 数据数组
   );
```

参数

*数组*

[in] 计算的数据数组。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

数组元素的平均绝对偏差。出错情况下它返回NaN（非数字）。

© 2000-2025, MetaQuotes Ltd.

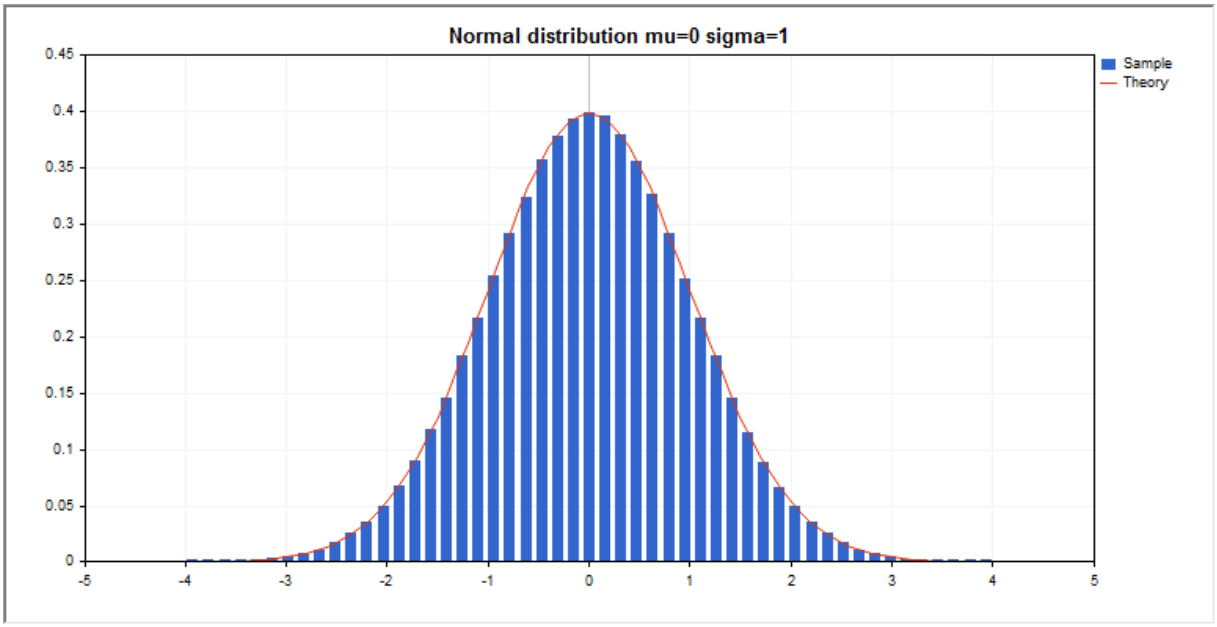
# 正态分布

这部分包含了处理正态分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据正态定律分布的伪随机数。该分布通过以下公式来定义：

$$f_{Normal}(x|\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

在这里：

- x —随机变量值
- μ —期望值
- σ —均方根偏差




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityNormal | 计算正态分布的概率密度函数 |
| MathCumulativeDistributionNormal | 计算正态概率分布函数值 |
| MathQuantileNormal | 计算指定概率逆正态分布函数值 |
| MathRandomNormal | 生成根据正态定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsNormal | 计算正态分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Normal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double mean_value=0;  // 期望值（平均数）
input double std_dev=1;     // 均方根偏差（标准偏差）
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得正态分布的样本
   MathRandomNormal(mean_value,std_dev,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNormal(x2,mean_value,std_dev,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Normal distribution mu=%G sigma=%G",mean_value
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
//--- 绘制全部曲线
   graphic.CurvePlotAll();
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
```

© 2000-2025, MetaQuotes Ltd.

```
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNormal

通过mu和sigma参数，计算随机变量x的正态分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNormal(
   const double  x,              // 随机变量值
   const double  mu,             // 分布的平均参数 (期望值)
   const double  sigma,          // 分布的sigma参数 (均方根偏差)
   const bool    log_mode,       // 计算数值对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的正态分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNormal(
   const double  x,              // 随机变量值
   const double  mu,             // 分布的平均参数 (期望值)
   const double  sigma,          // 分布的sigma参数 (均方根偏差)
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x[]的正态分布的概率密度函数值。出错情况下它返回false。R语言的dnorm()模拟。

```
bool  MathProbabilityDensityNormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 分布的平均参数 (期望值)
   const double   sigma,         // 分布的sigma参数 (均方根偏差)
   const bool     log_mode,      // 计算数值对数
   double&        result[]       //概率密度函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x[]的正态分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityNormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 分布的平均参数 (期望值)
   const double   sigma,         // 分布的sigma参数 (均方根偏差)
   double&        result[]       //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*mu*

[in] 分布的平均参数 (期望值)。

© 2000-2025, MetaQuotes Ltd.

*sigma*

[in] 分布的sigma参数（均方根偏差）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 获得概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNormal

通过mu和sigma参数，计算随机变量x的正态分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   const bool    tail,           // tail计算标识
   const bool    log_mode,       // 计算数值对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的正态分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x[]的正态分布函数值。出错情况下它返回false。R语言的dnorm()模拟。

```
bool  MathCumulativeDistributionNormal(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   const bool    tail,           // tail计算标识
   const bool    log_mode,       // 计算数值对数
   double&       result[]        // 概率函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x[]的正态分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionNormal(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*mu*

© 2000-2025, MetaQuotes Ltd.

[in] 分布的平均参数（期望值）。

*sigma*

[in] 分布的sigma参数（均方根偏差）。

*tail*

[in] 计算标识。如果tail=true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 获得概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileNormal

对于指定*probability*，函数通过mu和sigma参数计算逆正态分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNormal(
   const double  probability,    // 随机变量概率值/t5>
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   const bool    tail,           // tail计算标识
   const bool    log_mode,       // 计算数值对数
   int&          error_code      // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过mu和sigma参数计算逆正态分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNormal(
   const double  probability,    // 随机变量概率值/t5>
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   int&          error_code      // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过mu和sigma参数计算逆正态分布函数值。出错情况下它返回false。R语言的qnorm()模拟。

```
bool  MathQuantileNormal(
   const double& probability[],  // 随机变量概率值数组
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   const bool    tail,           // tail计算标识
   const bool    log_mode,       // 计算数值对数
   double&       result[]        // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过mu和sigma参数计算逆正态分布函数值。出错情况下它返回false。

```
bool  MathQuantileNormal(
   const double& probability[],  // 随机变量概率值数组
   const double  mu,             // 期望值
   const double  sigma,          // 均方根偏差
   double&       result[]        // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 分布的平均参数（期望值）。

*sigma*

[in] 分布的sigma参数（均方根偏差）。

*tail*

[in] 计算标识。如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 获得四分位数数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNormal

通过mu和sigma参数生成根据正态定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNormal(
   const double  mu,              // 期望值
   const double  sigma,           // 均方根偏差
   int&          error_code       // 存储错误代码的变量
   );
```

通过mu和sigma参数生成根据正态定律分布的伪随机变量。出错情况下它返回false。R语言的rnorm()模拟。

```
bool  MathRandomNormal(
   const double  mu,              // 期望值
   const double  sigma,           // 均方根偏差
   const int     data_count,      // 所需的数据量
   double&       result[]         // 获得伪随机变量的数组
   );
```

参数

*mu*

[in] 分布的平均参数（期望值）。

*sigma*

[in] 分布的sigma参数（均方根偏差）。

*data_count*

[in] 将获得的伪随机变量数。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNormal

计算正态分布的前4力矩的理论数值。

```
double  MathMomentsNormal(
   const double  mu,              // 期望值
   const double  sigma,           // 均方根偏差
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 存储错误代码的变量
   );
```

参数

*mu*

[in] 分布的平均参数（期望值）。

*sigma*

[in] 分布的sigma参数（均方根偏差）。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为False。

© 2000-2025, MetaQuotes Ltd.

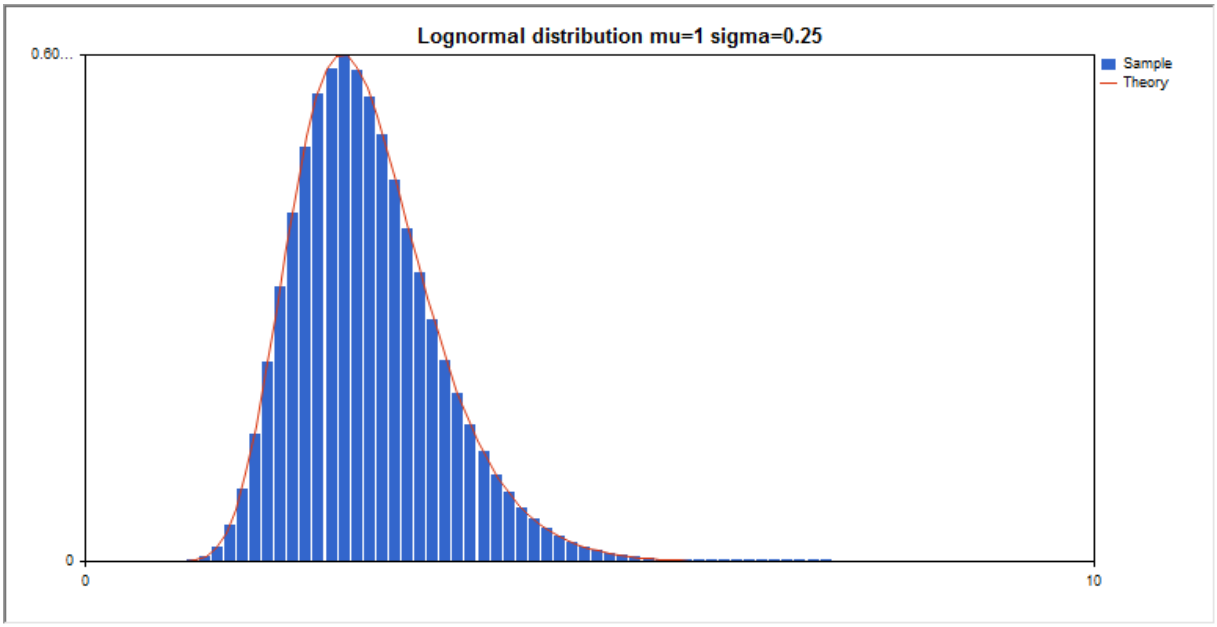
# 对数正态分布

这部分包含了处理对数正态分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据对数正态定律分布的伪随机数。对数正态分布通过以下公式来定义：

$$f_{Lognormal}(x|\mu,\sigma)=\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$$

在这里：

- x —随机变量值
- μ —期望值对数
- σ —均方根偏差对数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityLognormal | 计算对数正态分布的概率密度函数 |
| MathCumulativeDistributionLognormal | 计算对数正态概率分布函数值 |
| MathQuantileLognormal | 计算指定概率逆对数正态分布函数值 |
| MathRandomLognormal | 生成根据对数正态定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsLognormal | 计算对数正态分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Lognormal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double mean_value=1.0;  // 期望值对数 平均对数）
input double std_dev=0.25;    // 均方根偏差对数 对数标准偏差）
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得对数正态分布的样本
   MathRandomLognormal(mean_value,std_dev,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityLognormal(x2,mean_value,std_dev,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Lognormal distribution mu=%G sigma=%G",mean_va
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放Y轴
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(theor_max);
   graphic.YAxis().Min(0);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityLognormal

通过mu和sigma参数，计算随机变量x的对数正态分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityLognormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值对数（平均对数）
   const double  sigma,          // 均方根偏差对数（对数标准偏差）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的对数正态分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityLognormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值对数（平均对数）
   const double  sigma,          // 均方根偏差对数（对数标准偏差）
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x[]的对数正态分布的概率密度函数值。出错情况下它返回NaN。R语言的dlnorm()模拟。

```
bool  MathProbabilityDensityLognormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 期望值对数（平均对数）
   const double   sigma,         // 均方根偏差对数（对数标准偏差）
   const bool     log_mode,      //计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   double&        result[]       //概率密度函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x[]的对数正态分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityLognormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 期望值对数（平均对数）
   const double   sigma,         // 均方根偏差对数（对数标准偏差）
   double&        result[]       //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 期望值对数(log_mean)。

*sigma*

[in] 均方根偏差对数 (对数标准偏差)。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 获得概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionLognormal

通过mu和sigma参数，计算随机变量x的概率对数正态分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionLognormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值对数（平均对数）
   const double  sigma,          // 均方根偏差对数（对数标准偏差）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的概率对数正态分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionLognormal(
   const double  x,              // 随机变量值
   const double  mu,             // 期望值对数（平均对数）
   const double  sigma,          // 均方根偏差对数（对数标准偏差）
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x[]的概率对数正态分布函数。出错情况下它返回false。R语言的plnorm()模拟。

```
bool  MathCumulativeDistributionLognormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 期望值对数（平均对数）
   const double   sigma,         // 均方根偏差对数（对数标准偏差）
   const bool     tail,          // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool     log_mode,      // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对
   double&        result[]       // 概率函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x[]的概率对数正态分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionLognormal(
   const double&  x[],           // 随机变量值数组
   const double   mu,            // 期望值对数（平均对数）
   const double   sigma,         // 均方根偏差对数（对数标准偏差）
   double&        result[]       // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 期望值对数(log_mean)。

*sigma*

[in] 均方根偏差对数 (对数标准偏差)。

*tail*

[in] 计算标识，如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 获得概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileLognormal

对于指定概率，函数通过mu和sigma参数计算逆对数正态分布函数值。出错情况下它返回NaN。

```
double  MathQuantileLognormal(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 期望值对数（平均对数）
   const double  sigma,           // 均方根偏差对数（对数标准偏差）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率，函数通过mu和sigma参数计算逆对数正态分布函数值。出错情况下它返回NaN。

```
double  MathQuantileLognormal(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 期望值对数（平均对数）
   const double  sigma,           // 均方根偏差对数（对数标准偏差）
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的probability[]数组，函数通过mu和sigma参数计算逆对数正态分布函数值。出错情况下它返回false。R语言的qlnorm()模拟。

```
bool  MathQuantileLognormal(
   const double&  probability[],  // 随机变量概率值数组
   const double   mu,             // 期望值对数（平均对数）
   const double   sigma,          // 均方根偏差对数（对数标准偏差）
   const bool     tail,           //计算标识，如果false，则执行1.0-概率的计算
   const bool     log_mode,       // 计算标识，如果log_mode=true，则执行Exp(probability)计
   double&        result[]        // 四分位数值数组
   );
```

对于指定概率值的probability[]数组，函数通过mu和sigma参数计算逆对数正态分布函数值。出错情况下它返回false。

```
bool  MathQuantileLognormal(
   const double&  probability[],  // 随机变量概率值数组
   const double   mu,             // 期望值对数（平均对数）
   const double   sigma,          // 均方根偏差对数（对数标准偏差）
   double&        result[]        // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量出现的概率值。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 期望值对数(log_mean)。

*sigma*

[in] 均方根偏差对数 (对数标准偏差)。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomLognormal

通过mu和sigma参数生成根据对数正态定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomLognormal(
   const double  mu,              // 期望值对数 (平均对数)
   const double  sigma,           // 均方根偏差对数 (对数标准偏差)
   int&          error_code       // 存储错误代码的变量
   );
```

通过mu和sigma参数生成根据对数正态定律分布的伪随机变量。出错情况下它返回false。R语言的rlnorm()模拟。

```
double  MathRandomLognormal(
   const double  mu,              // 期望值对数 (平均对数)
   const double  sigma,           // 均方根偏差对数 (对数标准偏差)
   const int     data_count,      //所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*mu*

[in] 期望值对数(log_mean)。

*sigma*

[in] 均方根偏差对数 (对数标准偏差)。

*data_count*

[in] 所需的数据量。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 伪随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsLognormal

计算对数正态分布的前4力矩的理论数值如果力矩成功计算返回true，否则为false。

```
double  MathMomentsLognormal(
   const double  mu,              // 期望值对数 平均对数)
   const double  sigma,           // 均方根偏差对数 对数标准偏差)
   double&       mean,            // 平均值变量
   double&       variance,        //方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        //峰度变量
   int&          error_code       // 存储错误代码的变量
   );
```

参数

*mu*

[in]期望值对数(log_mean)。

*sigma*

[in] 均方根偏差对数 对数标准偏差)。

*平均值*

[in] 平均值变量。

*方差*

[out] 方差变量。

*偏度*

[out] 偏度变量。

*峰度*

[out] 峰度变量。

*错误代码*

[out] 存储错误代码的变量。

返回值

如果力矩成功计算返回true，否则为False。

© 2000-2025, MetaQuotes Ltd.

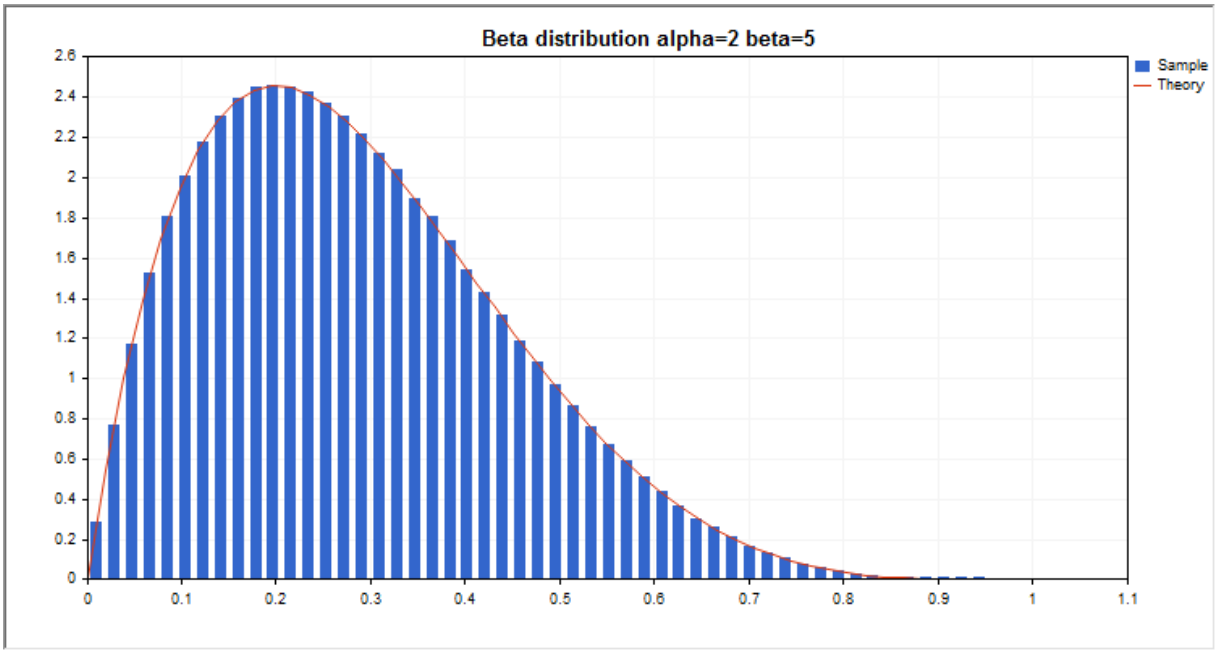
# Beta分布

这部分包含了处理Beta分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据相应定律分布的伪随机数。Beta分布通过以下公式来定义：

$$f_{Beta}(x|a,b)=\frac{1}{B(a,b)}x^{a-1}(1-x)^{b-1}$$

在这里：

- x —随机变量值
- a – Beta分布的第一个参数
- b —Beta分布的第二个参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityBeta | 计算Beta分布的概率密度函数 |
| MathCumulativeDistributionBeta | 计算Beta概率分布函数值 |
| MathQuantileBeta | 计算指定概率逆Beta分布函数值 |
| MathRandomBeta | 生成根据Beta分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsBeta | 计算Beta分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Beta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double alpha=2;    // Beta分布的第一个参数(shape1)
input double beta=5;     // Beta分布的第二个参数 (shape2)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//---获得beta分布的样本
   MathRandomBeta(alpha,beta,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityBeta(x2,alpha,beta,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Beta distribution alpha=%G beta=%G",alpha,beta
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
```

© 2000-2025, MetaQuotes Ltd.

```
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityBeta

通过a和b参数，计算随机变量x的Beta分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的Beta分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的Beta分布的概率密度函数值。出错情况下它返回false。R语言的dbeta()模拟。

```
bool  MathProbabilityDensityBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的Beta分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] Beta分布的第一个参数 (shape1)。

© 2000-2025, MetaQuotes Ltd.

*b*

[in] Beta分布的第二个参数 (shape2)

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionBeta

通过a和b参数，计算随机变量x的Beta分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const bool    tail,           //计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的Beta分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的Beta分布的概率分布函数。出错情况下它返回false。R语言的pbeta()模拟。

```
bool  MathCumulativeDistributionBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

<t0>通过a和b参数，计算随机变量数组x[]的Beta分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

© 2000-2025, MetaQuotes Ltd.

[in] Beta分布的第一个参数 (shape1)。

*b*

[in] Beta分布的第二个参数 (shape2)

*tail*

[in] 计算标识，如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileBeta

对于指定*probability*，函数通过a和b参数计算逆Beta分布函数值。出错情况下它返回NaN。

```
double  MathQuantileBeta(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过a和b参数计算逆Beta分布函数值。出错情况下它返回NaN。

```
double  MathQuantileBeta(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆Beta分布函数值。出错情况下它返回false。R语言的qbeta()模拟。

```
double  MathQuantileBeta(
   const double&  probability[], //随机变量概率值数组
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆Beta分布函数值。出错情况下它返回false。

```
bool  MathQuantileBeta(
   const double& probability[],  // 随机变量概率值数组
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*a*

[in] Beta分布的第一个参数(shape1)。

*b*

[in] Beta分布的第二个参数(shape2)。

*tail*

[in] 计算标识，如果lower_tail=false，则计算执行1.0-概率。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomBeta

通过a和b参数生成根据Beta分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomBeta(
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数生成根据Beta分布定律分布的伪随机变量。出错情况下它返回false。R语言的rbeta()模拟。

```
bool  MathRandomBeta(
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const int     data_count,      // 所需的数据量
   double&       result[]         // 获得伪随机变量的数组
   );
```

参数

*a*

[in] Beta分布的第一个参数(shape1)

*b*

[in] Beta分布的第二个参数(shape2)。

*data_count*

[in] 将获得的伪随机变量数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsBeta

计算Beta分布的前4力矩的理论数值。

```
double  MathMomentsBeta(
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] Beta分布的第一个参数(shape1)。

*b*

[in] Beta分布的第二个参数(shape2)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为False。

© 2000-2025, MetaQuotes Ltd.

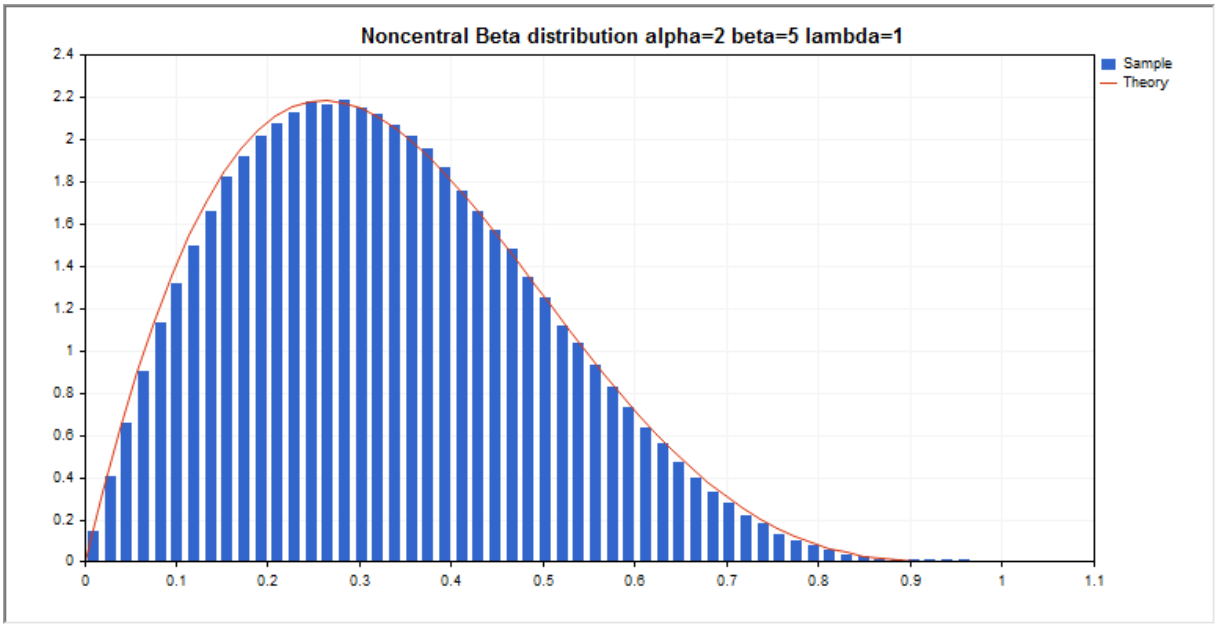
# 非中心beta分布

这部分包含了处理非中心Beta分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据相应定律分布的伪随机数。非中心Beta分布通过以下公式来定义：

$$f_{NoncentralBeta}(x|a,b,\lambda)=\sum_{r=0}^{\infty}e^{-\frac{\lambda}{2}}\frac{\left(\frac{\lambda}{2}\right)^{r}}{r!}\frac{x^{a+r-1}(1-x)^{b-1}}{B(a+r,b)}$$

在这里：

- x —随机变量值
- a – Beta分布的第一个参数
- b —Beta分布的第二个参数
- λ—非中心参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityNoncentralBeta | 计算非中心Beta分布的概率密度函数 |
| MathCumulativeDistributionNoncentralBeta | 计算非中心Beta概率分布函数值 |
| MathQuantileNoncentralBeta | 计算指定概率逆非中心Beta分布函数值 |
| MathRandomNoncentralBeta | 生成根据非中心Beta分布定律分布的伪随机变量/伪随机变量数组 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathMomentsNoncentralBeta | 计算非中心Beta分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralBeta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double a_par=2;     // Beta分布的第一个参数(shape1)
input double b_par=5;     // Beta分布的第二个参数 (shape2)
input double l_par=1;     // 非中心参数 (lambda)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=53;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得非中心beta分布的样本
   MathRandomNoncentralBeta(a_par,b_par,l_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralBeta(x2,a_par,b_par,l_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
```

© 2000-2025, MetaQuotes Ltd.

```
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral Beta distribution alpha=%G beta=%G
                          a_par,b_par,l_par));
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNoncentralBeta

通过a，b和lambda参数，计算随机变量x的非中心Beta分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a，b和lambda参数，计算随机变量x的非中心Beta分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a，b和lambda参数，计算随机变量数组x[]的非中心Beta分布的概率密度函数值。出错情况下它返回false。R语言的dbeta()模拟。

```
bool  MathProbabilityDensityNoncentralBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a，b和lambda参数，计算随机变量数组x[]的非中心Beta分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityNoncentralBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] Beta分布的第一个参数 (shape1)。

*b*

[in] Beta分布的第二个参数 (shape2)

*lambda*

[in] 非中心参数

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNoncentralBeta

通过a，b和lam bda参数，计算随机变量x的非中心Beta分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a，b和lam bda参数，计算随机变量x的非中心Beta分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralBeta(
   const double  x,              // 随机变量值
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a，b和lam bda参数，计算随机变量数组x[]的非中心Beta分布的概率分布函数。出错情况下它返回false。R语言的pbeta()模拟。

```
bool  MathCumulativeDistributionNoncentralBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过a，b和lam bda参数，计算随机变量数组x[]的非中心Beta分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionNoncentralBeta(
   const double& x[],            // 随机变量值数组
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   double&       result[]        // 概率函数值数组
   );
```

© 2000-2025, MetaQuotes Ltd.

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] Beta分布的第一个参数 (shape1)。

*b*

[in] Beta分布的第二个参数 (shape2)

*lambda*

[in] 非中心参数

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileNoncentralBeta

通过a，b和lam bda参数，计算随机变量*probability*出现概率的逆非中心Beta分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralBeta(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const double  lambda,          // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

通过a，b和lam bda参数，计算随机变量*probability*出现概率的逆非中心Beta分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralBeta(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const double  lambda,          // 非中心参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的probability□数组，函数通过a，b和lam bda参数计算逆非中心Beta分布的概率分布函数值。出错情况下它返回false。R语言的qbeta()模拟。

```
double  MathQuantileNoncentralBeta(
   const double&  probability[],  // 随机变量概率值数组
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const double  lambda,          // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的probability□数组，函数通过a，b和lam bda参数计算逆非中心Beta分布的概率分布函数值。出错情况下它返回false。

```
bool  MathQuantileNoncentralBeta(
   const double& probability[],  // 随机变量概率值数组
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const double  lambda,          // 非中心参数
   double&       result[]         // 四分位数值数组
   );
```

© 2000-2025, MetaQuotes Ltd.

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*a*

[in] Beta分布的第一个参数(shape1)。

*b*

[in] Beta分布的第二个参数(shape2)。

*lambda*

[in] 非中心参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNoncentralBeta

通过a，b和lambda参数生成根据非中心Beta分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNoncentralBeta(
   const double  a,              // Beta分布的第一个参数(shape1)
   const double  b,              // Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a，b和lambda参数生成根据非中心Beta分布定律分布的伪随机变量。出错情况下它返回false。R语言的rbeta()模拟。

```
bool  MathRandomNoncentralBeta(
   const double  a,               //Beta分布的第一个参数 (shape1)
   const double  b,               //Beta分布的第二个参数 (shape2)
   const double  lambda,          // 非中心参数
   const int     data_count,      // 所需的数据量
   double&       result[]         // 获得伪随机变量的数组
   );
```

参数

*a*

[in] Beta分布的第一个参数(shape1)

*b*

[in] Beta分布的第二个参数(shape2)。

*lambda*

[in] 非中心参数

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNoncentralBeta

通过a，b和lam bda参数计算非中心Beta分布的前4力矩的理论数值。

```
double  MathMomentsNoncentralBeta(
   const double  a,              //Beta分布的第一个参数 (shape1)
   const double  b,              //Beta分布的第二个参数 (shape2)
   const double  lambda,         // 非中心参数
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] Beta分布的第一个参数(shape1)。

*b*

[in] Beta分布的第二个参数(shape2)。

*lambda*

[in] 非中心参数

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

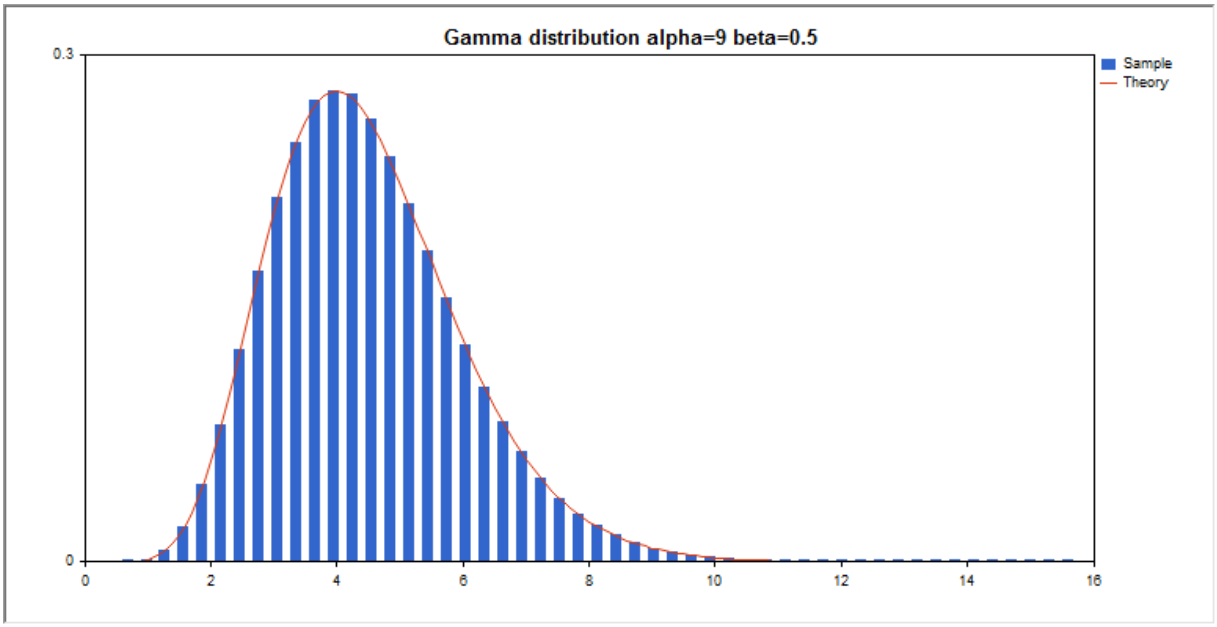
# 伽玛分布

这部分包含了处理伽玛分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据相应定律分布的伪随机数。伽玛分布通过以下公式来定义：

$$f_{Gamma}(x|a,b)=\frac{1}{b^{a}\Gamma(a)}x^{a-1}e^{-\frac{x}{b}}$$

在这里：

- x —随机变量值
- a – 分布的第一个参数
- b —分布的第二个参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityGamma | 计算伽玛分布的概率密度函数 |
| MathCumulativeDistributionGamma | 计算伽玛概率分布函数值 |
| MathQuantileGamma | 计算指定概率逆伽玛分布函数值 |
| MathRandomGamma | 生成根据伽玛分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsGamma | 计算伽玛分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Gamma.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double alpha=9;   // 分布的第一个参数(shape)
input double beta=0.5;  // 分布的第二个参数(scale)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;        // 样本值的数量
   int ncells=51;        // 直方图区间数
   double x[];           // 直方图的区间中心
   double y[];           // 区间内样本数量值下降
   double data[];        // 随机值样本
   double max,min;       // 样本的最大最小值
//--- 获得伽玛分布的样本
   MathRandomGamma(alpha,beta,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityGamma(x2,alpha,beta,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Gamma distribution alpha=%G beta=%G",alpha,bet
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放Y轴
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(NormalizeDouble(theor_max,1));
   graphic.YAxis().Min(0);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityGamma

通过a和b参数，计算随机变量x的伽玛分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityGamma(
   const double  x,              // 随机变量值
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数(scale)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的伽玛分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityGamma(
   const double  x,              // 随机变量值
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的伽玛分布的概率密度函数值。出错情况下它返回false。R语言的dgamma()模拟。

```
bool  MathProbabilityDensityGamma(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数 (scale)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的伽玛分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityGamma(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数 (scale)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] 分布的第一个参数(shape)。

© 2000-2025, MetaQuotes Ltd.

*b*

[in] 分布的第二个参数(scale)。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionGamma

通过a和b参数，计算随机变量x的伽玛分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionGamma(
   const double  x,              // 随机变量值
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数(scale)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的伽玛分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionGamma(
   const double  x,              // 随机变量值
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的伽玛分布的概率分布函数。出错情况下它返回false。R语言的pgamma()模拟。

```
bool  MathCumulativeDistributionGamma(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数 (scale)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的伽玛分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionGamma(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数 (scale)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

© 2000-2025, MetaQuotes Ltd.

[in] 分布的第一个参数(shape)。

*b*

[in] 分布的第二个参数(scale)

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileGamma

对于指定*probability*，函数通过a和b参数计算逆伽玛分布函数值。出错情况下它返回NaN。

```
double  MathQuantileGamma(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布的第一个参数(shape)
   const double  b,               // 分布的第二个参数 (scale)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过a和b参数计算逆伽玛分布函数值。出错情况下它返回NaN。

```
double  MathQuantileGamma(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布的第一个参数(shape)
   const double  b,               // 分布的第二个参数 (scale)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆伽玛分布函数值。出错情况下它返回false。R语言的qgamma()模拟。

```
double  MathQuantileGamma(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布的第一个参数(shape)
   const double  b,               // 分布的第二个参数 (scale)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆伽玛分布函数值。出错情况下它返回false。

```
bool  MathQuantileGamma(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布的第一个参数(shape)
   const double  b,               // 分布的第二个参数 (scale)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*a*

[in] 分布的第一个参数(shape)。

*b*

[in] 分布的第二个参数(scale)。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomGamma

通过a和b参数生成根据伽玛分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomGamma(
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数生成根据伽玛分布定律分布的伪随机变量。出错情况下它返回false。R语言的rgamma()模拟。

```
bool  MathRandomGamma(
   const double  a,               // 分布的第一个参数(shape)
   const double  b,               // 分布的第二个参数 (scale)
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*a*

[in] 分布的第一个参数(shape)。

*b*

[in] 分布的第二个参数(scale)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsGamma

通过a和b参数计算伽玛分布的前4力矩的理论数值。

```
double  MathMomentsGamma(
   const double  a,              // 分布的第一个参数(shape)
   const double  b,              // 分布的第二个参数 (scale)
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] 分布的第一个参数(shape)。

*b*

[in] 分布的第二个参数(scale)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

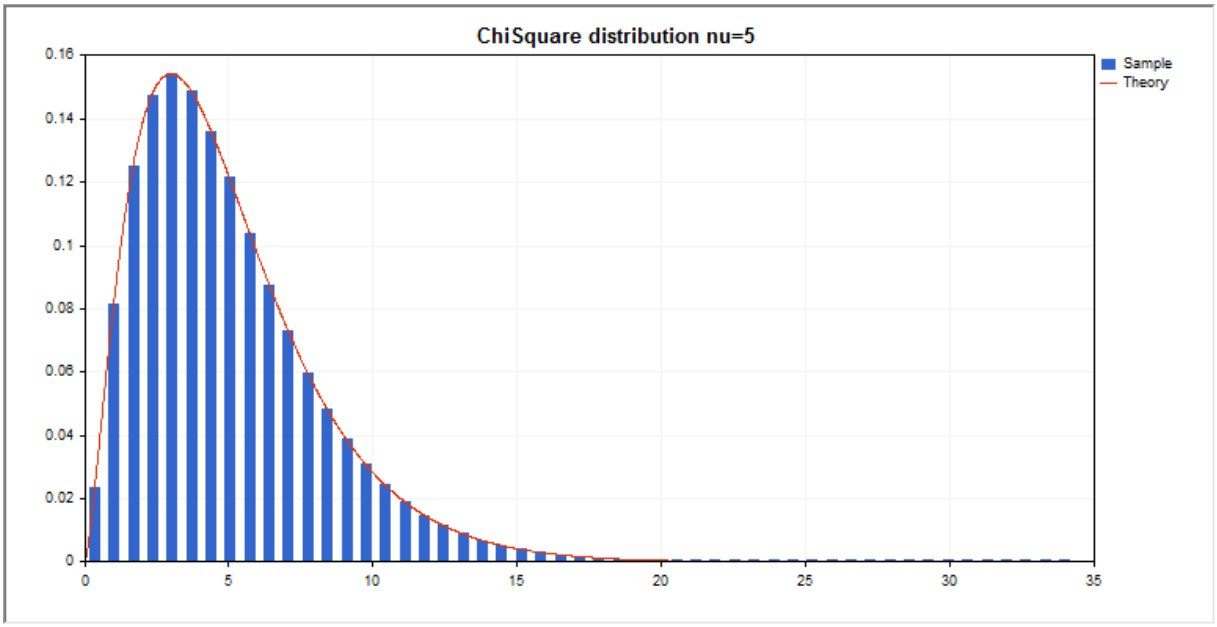
# 卡方分布

这部分包含了处理卡方分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据相应定律分布的伪随机数。卡方分布通过以下公式来定义：

$$f_{Chi-Square}(x|\nu)=\frac{x^{\frac{(\nu-2)}{2}}e^{-\frac{x}{2}}}{2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)}$$

在这里：

- x —随机变量值
- ν —自由度数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityChiSquare | 计算卡方分布的概率密度函数 |
| MathCumulativeDistributionChiSquare | 计算卡方概率分布函数值 |
| MathQuantileChiSquare | 计算指定概率逆卡方分布函数值 |
| MathRandomChiSquare | 生成根据卡方分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsChiSquare | 计算卡方分布的前4力矩的理论数值 |

© 2000-2025, MetaQuotes Ltd.

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\ChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_par=5;     // 自由度数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得卡方分布的样本
   MathRandomChiSquare(nu_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityChiSquare(x2,nu_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
```

© 2000-2025, MetaQuotes Ltd.

```
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("ChiSquare distribution nu=%G ",nu_par));
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
```

© 2000-2025, MetaQuotes Ltd.

```
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityChiSquare

通过nu参数，计算随机变量x的卡方分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量x的卡方分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量数组x[]的卡方分布的概率密度函数值。出错情况下它返回false。R语言的dchisq()模拟。

```
bool  MathProbabilityDensityChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过nu参数，计算随机变量数组x[]的卡方分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu*

[in] 分布参数 (自由度数)

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

© 2000-2025, MetaQuotes Ltd.

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionChiSquare

通过nu参数，计算随机变量x的卡方分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量x的卡方分布的概率分布函数。出错情况下它返回NaN

```
double  MathCumulativeDistributionChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量数组x[]的卡方分布的概率分布函数。出错情况下它返回false。R语言的pchisq()模拟。

```
bool  MathCumulativeDistributionChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过nu参数，计算随机变量数组x[]的卡方分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu*

[in] 分布参数 (自由度数)。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileChiSquare

对于指定*probability*，函数计算逆卡方分布函数值。出错情况下它返回NaN。

```
double  MathQuantileChiSquare(
   const double  probability,    // 随机变量出现的概率值
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,       //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code      // 存储错误代码的变量
   );
```

对于指定*probability*，函数计算逆卡方分布函数值。出错情况下它返回NaN。

```
double  MathQuantileChiSquare(
   const double  probability,    // 随机变量出现的概率值
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]* 数组，函数计算逆卡方分布函数值。出错情况下它返回false。R语言的qchisq()模拟。

```
double  MathQuantileChiSquare(
   const double& probability[],  // 随机变量概率值数组
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,       //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]        // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数计算逆卡方分布函数值。出错情况下它返回false。

```
bool  MathQuantileChiSquare(
   const double& probability[],  // 随机变量概率值数组
   const double  nu,             // 分布参数 (自由度数)
   double&       result[]        // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*nu*

[in] 分布参数 (自由度数)。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomChiSquare

通过nu参数生成根据卡方分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomChiSquare(
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数生成根据卡方分布定律分布的伪随机变量。出错情况下它返回false。R语言的rchisq()模拟。

```
bool  MathRandomChiSquare(
   const double  nu,              // 分布参数 (自由度数)
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsChiSquare

通过nu参数计算卡方分布的前4力矩的理论数值。

```
double  MathMomentsChiSquare(
   const double  nu,              // 分布参数 (自由度数)
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

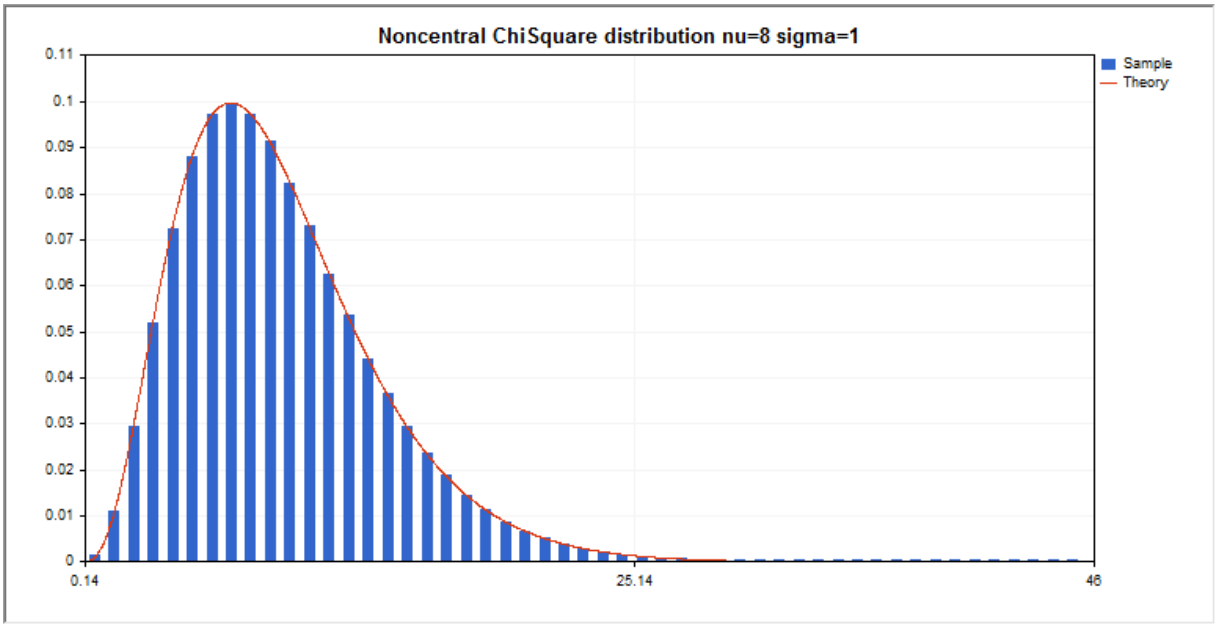
# 非中心卡方分布

这部分包含了处理非中心卡方分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据相应定律分布的伪随机数。非中心卡方分布通过以下公式来定义：

$$f_{NoncentralChiSquare}(x|\nu,\sigma)=\frac{1}{2^{\frac{\nu}{2}}\Gamma\left(\frac{1}{2}\right)}x^{\frac{\nu}{2}-1}e^{-\frac{(x+\sigma)}{2}}\sum_{r=0}^{\infty}\frac{(\lambda x)^r}{(2r)!}\frac{\Gamma\left(\frac{1}{2}+r\right)}{\Gamma\left(\frac{\nu}{2}+r\right)}$$

在这里：

- x — 随机变量值
- ν — 自由度数
- σ — 非中心参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityNoncentralChiSquare | 计算非中心卡方分布的概率密度函数 |
| MathCumulativeDistributionNoncentralChiSquare | 计算非中心卡方概率分布函数值 |
| MathQuantileNoncentralChiSquare | 计算指定概率逆非中心卡方分布函数值 |
| MathRandomNoncentralChiSquare | 生成根据非中心卡方分布定律分布的伪随机变量/伪随机变量数组 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathMomentsNoncentralChiSquare | 计算非中心卡方分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_par=8;     // 自由度数
input double si_par=1;     // 非中心参数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得非中心卡方分布的样本
   MathRandomNoncentralChiSquare(nu_par,si_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralChiSquare(x2,nu_par,si_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
```

© 2000-2025, MetaQuotes Ltd.

```
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral ChiSquare distribution nu=%G sigma=
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放X轴
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(NormalizeDouble(max,0));
   graphic.XAxis().Min(min);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
```

© 2000-2025, MetaQuotes Ltd.

```
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNoncentralChiSquare

通过nu和sigma参数，计算随机变量x的非中心卡方分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和sigma参数，计算随机变量x的非中心卡方分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和sigma参数，计算随机变量数组x[]的非中心卡方分布的概率密度函数值。出错情况下它返回false。R语言的dchisq()模拟。

```
bool  MathProbabilityDensityNoncentralChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu parameter for an array of random variables x[]. 出错情况下它返回false。

```
bool  MathProbabilityDensityNoncentralChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*nu*

[in] 分布参数（自由度数）。

*sigma*

[in] 非中心参数。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNoncentralChiSquare

通过nu和sigma参数，计算随机变量x的非中心卡方分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   const bool    tail,           //计算标识，如果lower_tail=true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和sigma参数，计算随机变量x的非中心卡方分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralChiSquare(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和sigma参数，计算随机变量数组x[]的非中心卡方分布的概率分布函数。出错情况下它返回false。R语言的pchisq()模拟。

```
bool  MathCumulativeDistributionNoncentralChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   const bool    tail,           // 计算标识，如果lower_tail=true，则计算不超过随机变量x的概
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对
   double&       result[]        // 概率函数值数组
   );
```

通过nu和sigma参数，计算随机变量数组x[]的非中心卡方分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionNoncentralChiSquare(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值数组。

*nu*

[in] 分布参数（自由度数）。

*sigma*

[in] 非中心参数。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileNoncentralChiSquare

对于指定*probability*，函数通过nu和sigma参数计算逆非中心卡方分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralChiSquare(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数（自由度数）
   const double  sigma,           // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过nu和sigma参数计算逆非中心卡方分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralChiSquare(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数（自由度数）
   const double  sigma,           // 非中心参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过nu和sigma参数计算逆非中心卡方分布函数值。出错情况下它返回false。R语言的qchisq()模拟。

```
double  MathQuantileNoncentralChiSquare(
   const double&  probability[],  // 随机变量概率值数组
   const double  nu,              // 分布参数（自由度数）
   const double  sigma,           // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        // 计算标识，如果log_mode=true，则执行Exp(probability)计
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数计算逆非中心卡方分布函数值。出错情况下它返回false。

```
bool  MathQuantileNoncentralChiSquare(
   const double& probability[],  // 随机变量概率值数组
   const double  nu,             // 分布参数（自由度数）
   const double  sigma,          // 非中心参数
   double&       result[]        // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*nu*

[in] 分布参数（自由度数）。

*sigma*

[in] 非中心参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNoncentralChiSquare

通过nu和sigma参数生成根据非中心卡方分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNoncentralChiSquare(
   const double  nu,             // 分布参数 (自由度数)
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和sigma参数生成根据非中心卡方分布定律分布的伪随机变量。出错情况下它返回false。R语言的rchisq()模拟。

```
bool  MathRandomNoncentralChiSquare(
   const double  nu,              // 分布参数 (自由度数)
   const double  sigma,           // 非中心参数
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*sigma*

[in] 非中心参数。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNoncentralChiSquare

通过nu和sigma参数计算非中心卡方分布的前4力矩的理论数值。

```
double  MathMomentsNoncentralChiSquare(
   const double  nu,              // 分布参数 (自由度数)
   const double  sigma,           // 非中心参数
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*sigma*

[in] 非中心参数。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

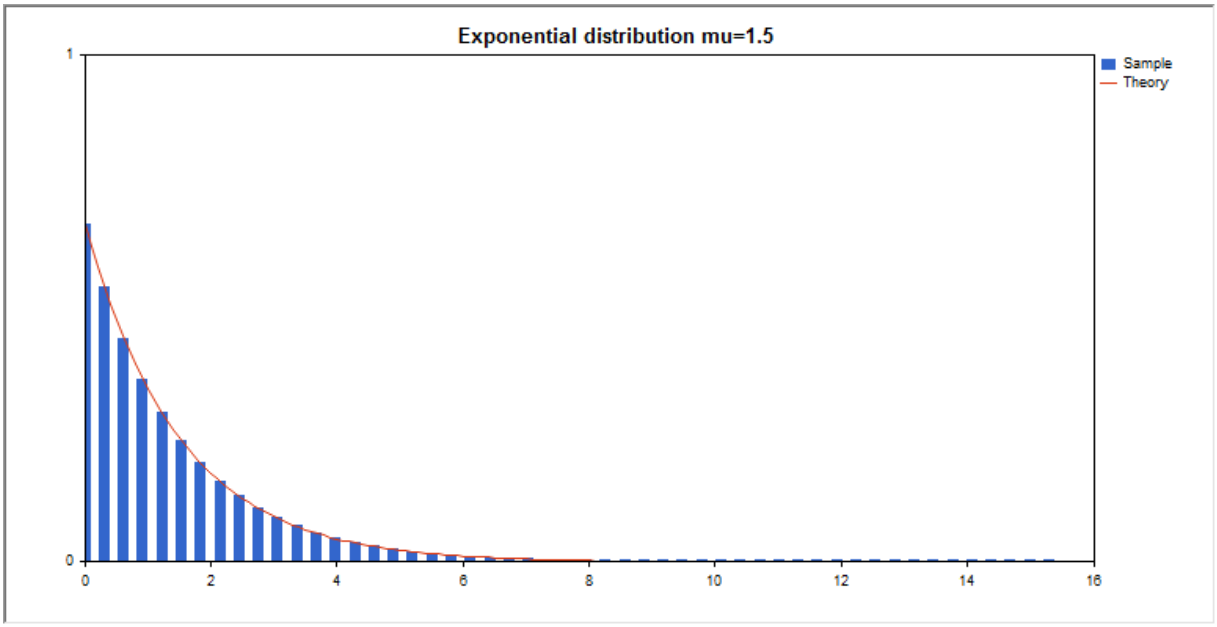
# 指数分布

这部分包含了处理指数分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据指数分布定律分布的伪随机数。指数分布通过以下公式来定义：

$$f_{Exponential}(x|\mu)=\frac{1}{\mu}e^{-\frac{x}{\mu}}$$

在这里：

- x —随机变量值
- μ —期望值




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityExponential | 计算指数分布的概率密度函数 |
| MathCumulativeDistributionExponential | 计算指数概率分布函数值 |
| MathQuantileExponential | 计算指定概率逆指数分布函数值 |
| MathRandomExponential | 生成根据指数分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsExponential | 计算指数分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\Exponential.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double mu_par=1.5;     // 自由度数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得指数分布的样本
   MathRandomExponential(mu_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityExponential(x2,mu_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Exponential distribution mu=%G ",mu_par));
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
```

```
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityExponential

通过mu参数，计算随机变量x的指数分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityExponential(
   const double  x,              // 随机变量值
   const double  mu,             // 分布参数 期望值）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu参数，计算随机变量x的指数分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityExponential(
   const double  x,              // 随机变量值
   const double  mu,             // 分布参数 期望值）
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu参数，计算随机变量数组x[]的指数分布的概率密度函数值。出错情况下它返回false。R语言的dexp()模拟。

```
bool  MathProbabilityDensityExponential(
   const double& x[],            // 随机变量值数组
   const double  mu,             //分布参数 期望值）
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过mu参数，计算随机变量数组x[]的指数分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityExponential(
   const double& x[],            // 随机变量值数组
   const double  mu,             //分布参数 期望值）
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*mu*

[in] 分布参数 期望值）

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

© 2000-2025, MetaQuotes Ltd.

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionExponential

通过mu参数，计算随机变量x的概率指数分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionExponential(
   const double  x,              // 随机变量值
   const double  mu,             // 分布参数 期望值）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu参数，计算随机变量x的概率指数分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionExponential(
   const double  x,              // 随机变量值
   const double  mu,             // 分布参数 期望值）
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu参数，计算随机变量数组x[]的概率指数分布函数。出错情况下它返回false。R语言的pexp()模拟。

```
bool  MathCumulativeDistributionExponential(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布参数 期望值）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过mu参数，计算随机变量数组x[]的概率指数分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionExponential(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布参数 期望值）
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*mu*

[in] 分布参数 (期望值)。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileExponential

对于指定*probability*，函数通过mu参数计算逆指数分布函数值。出错情况下它返回NaN。

```
double  MathQuantileExponential(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 分布参数 期望值）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过mu参数计算逆指数分布函数值。出错情况下它返回NaN。

```
double  MathQuantileExponential(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 分布参数 期望值）
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过mu参数计算逆指数分布函数值。出错情况下它返回false。R语言的qexp()模拟。

```
double  MathQuantileExponential(
   const double& probability[],   // 随机变量概率值数组
   const double  mu,              // 分布参数 期望值）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过mu参数计算逆指数分布函数值。出错情况下它返回false。

```
bool  MathQuantileExponential(
   const double& probability[],   // 随机变量概率值数组
   const double  mu,              // 分布参数 期望值）
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*mu*

[in] 分布参数（期望值）。

*tail*

© 2000-2025, MetaQuotes Ltd.

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomExponential

通过mu参数生成根据指数分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomExponential(
   const double  mu,             // 分布参数 期望值）
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu参数生成根据指数分布定律分布的伪随机变量。出错情况下它返回false。R语言的rexp()模拟。

```
bool  MathRandomExponential(
   const double  mu,              // 分布参数 期望值）
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*mu*

[in] 分布参数（期望值）。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsExponential

通过mu参数计算指数分布的前4力矩的理论数值。

```
double  MathMomentsExponential(
   const double  mu,              // 分布参数 期望值）
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*mu*

[in] 分布参数 期望值）。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

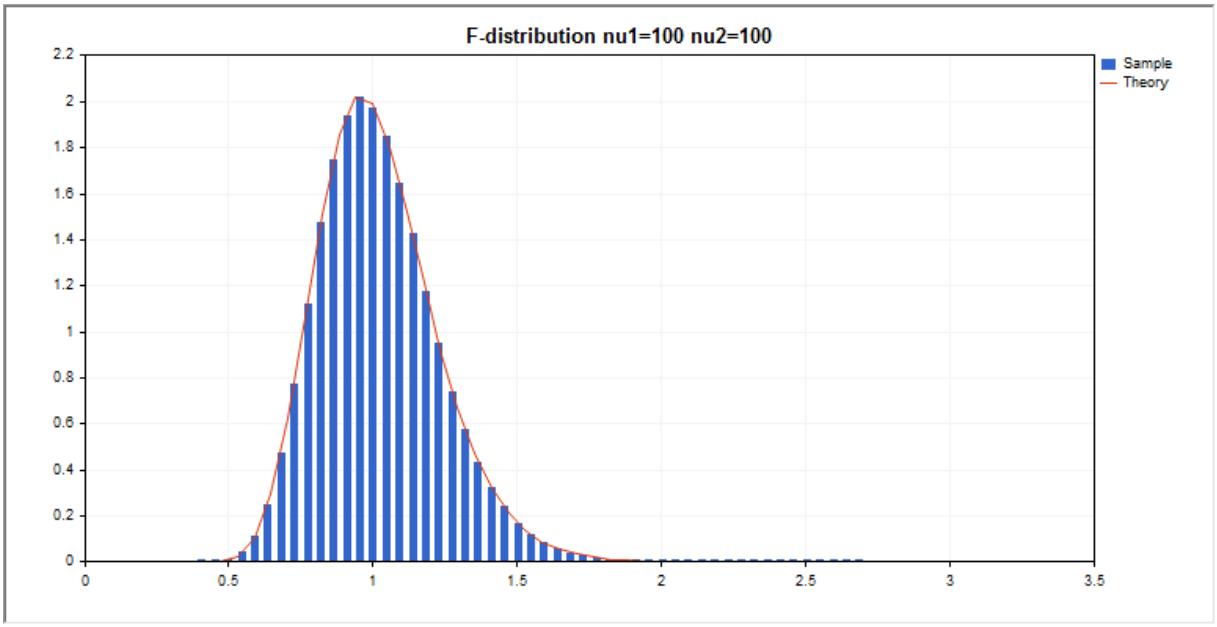
# F-分布

这部分包含了处理F-分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据费舍尔F-分布定律分布的伪随机数。F-分布通过以下公式来定义：

$$f_F(x|\nu_1,\nu_2)=\frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}\right)}{\Gamma\left(\frac{\nu_1}{2}\right)\Gamma\left(\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}}\frac{x^{\frac{\nu_1-2}{2}}}{\left(1+\left(\frac{\nu_1}{\nu_2}\right)x\right)^{\frac{\nu_1+\nu_2}{2}}}$$

在这里：

- x —随机变量值
- ν1 —分布的第一个参数（自由度数）
- ν2 —分布的第二个参数（自由度数）




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
|---|---|
| MathProbabilityDensityF | 计算F-分布的概率密度函数 |
| MathCumulativeDistributionF | 计算F-分布函数值 |
| MathQuantileF | 计算指定概率逆F-分布函数值 |
| MathRandomF | 生成根据费舍尔定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsF | 计算费舍尔F-分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\F.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_1=100;     //第一个自由度数
input double nu_2=100;     // 第二个自由度数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得费舍尔F-分布的样本
   MathRandomF(nu_1,nu_2,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityF(x2,nu_1,nu_2,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("F-distribution nu1=%G nu2=%G",nu_1,nu_2));
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(4);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
```

© 2000-2025, MetaQuotes Ltd.

```
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityF

通过nu1和nu2参数，计算随机变量x的费舍尔F-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1和nu2参数，计算随机变量x的费舍尔F-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1和nu2参数，计算随机变量数组x[]的费舍尔F-分布的概率密度函数值。出错情况下它返回false。R语言的df()模拟。

```
bool  MathProbabilityDensityF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过nu1和nu2参数，计算随机变量数组x[]的费舍尔F-分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu1*

[in] 分布的第一个参数 (自由度数)。

© 2000-2025, MetaQuotes Ltd.

*nu2*

[in] 分布的第二个参数（自由度数）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionF

通过nu1和nu2参数，计算随机变量x的费舍尔F-分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数，如果log_mode=true，则返回概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1和nu2参数，计算随机变量x的费舍尔F-分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1和nu2参数，计算随机变量x□的费舍尔F-分布的概率分布函数值。出错情况下它返回false。R语言的pf()模拟。

```
bool  MathCumulativeDistributionF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则返回概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过nu1和nu2参数，计算随机变量x□的费舍尔F-分布的概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*nu1*

[in] 分布的第一个参数（自由度数）。

*nu2*

[in] 分布的第二个参数（自由度数）。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileF

对于指定*probability*，函数通过nu1和nu2参数计算逆费舍尔F-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileF(
   const double  probability,     // 随机变量出现的概率值
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过nu1和nu2参数计算逆费舍尔F-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileF(
   const double  probability,     // 随机变量出现的概率值
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过nu1和nu2参数计算逆费舍尔F-分布函数值。出错情况下它返回false。R语言的qf()模拟。

```
double  MathQuantileF(
   const double& probability[],   // 随机变量概率值数组
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过nu1和nu2参数计算逆费舍尔F-分布函数值。出错情况下它返回false。

```
bool  MathQuantileF(
   const double& probability[],   // 随机变量概率值数组
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*tail*

[in] 计算标识，如果lower_tail=false，则计算执行1.0-概率。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomF

通过nu1和nu2参数生成根据费舍尔F-分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1和nu2参数生成根据费舍尔F-分布定律分布的伪随机变量。出错情况下它返回false。R语言的rf()模拟。

```
bool  MathRandomF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsF

通过nu1和nu2参数计算费舍尔F-分布的前4力矩的理论数值。

```
double  MathMomentsF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

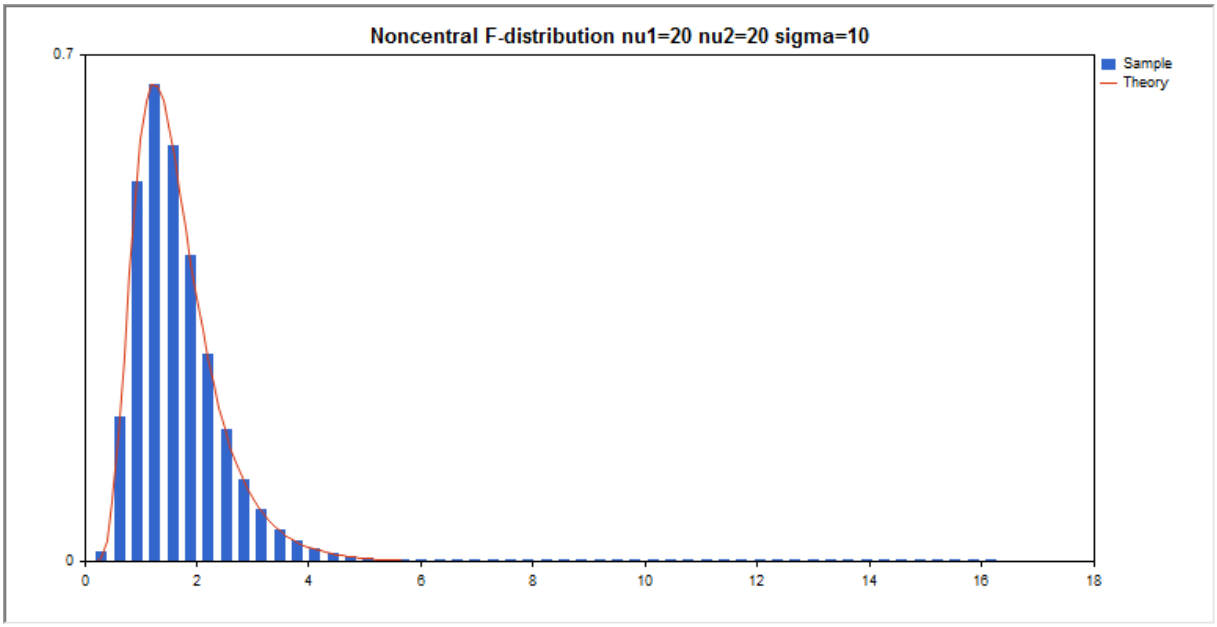
# 非中心F-分布

这部分包含了处理非中心F-分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据非中心费舍尔的F-分布定律分布的伪随机数。非中心F-分布通过以下公式来定义：

$$f_{NoncentralF}(x|\nu_1,\nu_2,\sigma)=e^{-\frac{\sigma}{2}}\sum_{r=0}^{\infty}\frac{1}{r!}\left(\frac{\sigma}{2}\right)^r\frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}+r\right)}{\Gamma\left(\frac{\nu_2}{2}+r\right)\Gamma\left(\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_2}{2}+r}\frac{x^{\frac{\nu_2}{2}-1+r}}{\left(1+\frac{\nu_1}{\nu_2}x\right)^{\frac{\nu_1+\nu_2}{2}+r}}$$

在这里：

- x —随机变量值
- ν1 —分布的第一个参数（自由度数）
- ν2 —分布的第二个参数（自由度数）
- σ —非中心参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
|---|---|
| MathProbabilityDensityNoncentralF | 计算非中心F-分布的概率密度函数 |
| MathCumulativeDistributionNoncentralF | 计算非中心F-分布函数值 |
| MathQuantileNoncentralF | 计算指定概率逆非中心F-分布函数值 |
| MathRandomNoncentralF | 生成根据非中心费舍尔分布定律分布的伪随机变量/伪随机变量数组 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathMomentsNoncentralF | 计算非中心费舍尔F-分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralF.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_1=20;     // 第一个自由度数
input double nu_2=20;     // 第二个自由度数
input double sig=10;      // 非中心参数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得非中心费舍尔F-分布的样本
   MathRandomNoncentralF(nu_1,nu_2,sig,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralF(x2,nu_1,nu_2,sig,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
```

© 2000-2025, MetaQuotes Ltd.

```
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral F-distribution nu1=%G nu2=%G sigma=
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNoncentralF

通过nu1，nu2和sigma参数，计算随机变量x的非中心费舍尔F-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1，nu2和sigma参数，计算随机变量x的非中心费舍尔F-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1，nu2和sigma参数，计算随机变量数组x[]的非中心费舍尔F-分布的概率密度函数值。出错情况下它返回false。R语言的df()模拟。

```
bool  MathProbabilityDensityNoncentralF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过nu1，nu2和sigma参数，计算随机变量数组x[]的非中心费舍尔F-分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityNoncentralF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*sigma*

[in] 非中心参数。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNoncentralF

通过nu1，nu2和sigma参数，计算随机变量x的非中心费舍尔F-分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1，nu2和sigma参数，计算随机变量x的非中心费舍尔F-分布的概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralF(
   const double  x,              // 随机变量值
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1，nu2和sigma参数，计算随机变量数组x[]的非中心费舍尔F-分布的概率分布函数值。出错情况下它返回false。R语言的pf()模拟。

```
bool  MathCumulativeDistributionNoncentralF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过nu1，nu2和sigma参数，计算随机变量数组x[]的非中心费舍尔F-分布的概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionNoncentralF(
   const double& x[],            // 随机变量值数组
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   double&       result[]        // 概率函数值数组
   );
```

© 2000-2025, MetaQuotes Ltd.

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*sigma*

[in] 非中心参数。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileF

对于指定*probability*，函数通过nu1，nu2和sigma参数计算逆非中心费舍尔F-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileF(
   const double  probability,     // 随机变量出现的概率值
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   const double  sigma,           // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过nu1，nu2和sigma参数计算逆非中心费舍尔F-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileF(
   const double  probability,     // 随机变量出现的概率值
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   const double  sigma,           // 非中心参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过nu1，nu2和sigma参数计算逆非中心费舍尔F-分布函数值。出错情况下它返回false。R语言的qf()模拟。

```
double  MathQuantileF(
   const double& probability[],   // 随机变量概率值数组
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   const double  sigma,           // 非中心参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过nu1，nu2和sigma参数计算逆非中心费舍尔F-分布函数值。出错情况下它返回false。

```
bool  MathQuantileF(
   const double& probability[],   // 随机变量概率值数组
   const double  nu1,             // 分布的第一个参数 (自由度数)
   const double  nu2,             // 分布的第二个参数 (自由度数)
   double&       result[]         // 四分位数值数组
   );
```

参数

© 2000-2025, MetaQuotes Ltd.

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*sigma*

[in] 非中心参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNoncentralF

通过nu1，nu2和sigma参数生成根据非中心费舍尔F-分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNoncentralF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu1，nu2和sigma参数生成根据非中心费舍尔F-分布定律分布的伪随机变量。出错情况下它返回false。R语言的rf()模拟。

```
bool  MathRandomNoncentralF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*sigma*

[in] 非中心参数。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNoncentralF

通过nu1，nu2和sigma参数计算非中心费舍尔F-分布的前4力矩的理论数值。

```
double  MathMomentsNoncentralF(
   const double  nu1,            // 分布的第一个参数 (自由度数)
   const double  nu2,            // 分布的第二个参数 (自由度数)
   const double  sigma,          // 非中心参数
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*nu1*

[in] 分布的第一个参数 (自由度数)。

*nu2*

[in] 分布的第二个参数 (自由度数)。

*sigma*

[in] 非中心参数。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

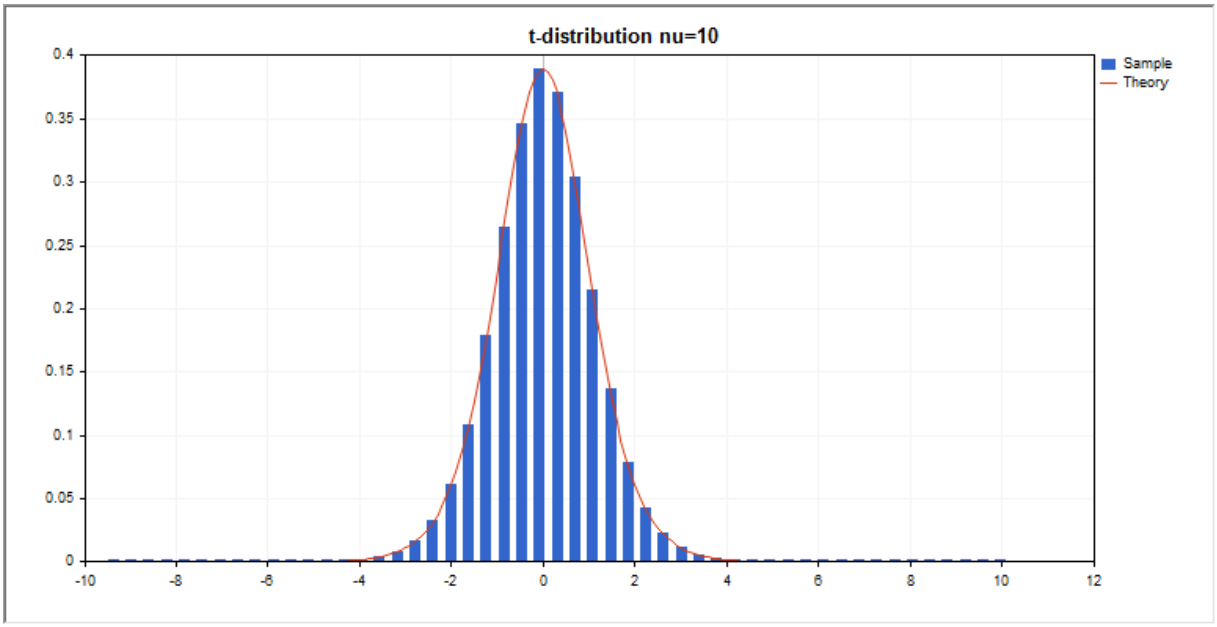
# t-分布

这部分包含了处理学生t-分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据学生定律分布的伪随机数。t-分布通过以下公式来定义：

$$f_T(x|\nu)=\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)}\frac{1}{\sqrt{\pi\nu}}\frac{1}{\left(1+\frac{x^2}{\nu}\right)^{\frac{\nu+1}{2}}}$$

在这里：

- x —随机变量值
- ν — 分布参数（自由度数）




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
|---|---|
| MathProbabilityDensityT | 计算t-分布的概率密度函数 |
| MathCumulativeDistributionT | 计算t-分布函数值 |
| MathQuantileT | 计算指定概率逆t-分布函数值 |
| MathRandomT | 生成根据学生t-分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsT | 计算学生t-分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\T.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_par=10;    // 自由度数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;        // 样本值的数量
   int ncells=51;        // 直方图区间数
   double x[];           // 直方图的区间中心
   double y[];           // 区间内样本数量值下降
   double data[];        // 随机值样本
   double max,min;       // 样本的最大最小值
//--- 获得学生t-分布的样本
   MathRandomT(nu_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityT(x2,nu_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("t-distribution nu=%G",nu_par));
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
```

© 2000-2025, MetaQuotes Ltd.

```
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityT

通过nu参数，计算随机变量x的学生t-分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量x的学生t-分布的概率密度函数的值。出错情况下它返回NaN。

```
double  MathProbabilityDensityT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量数组x[]的学生t-分布的概率密度函数值。出错情况下它返回false。R语言的dt()模拟。

```
bool  MathProbabilityDensityT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过nu参数，计算随机变量数组x[]的学生t-分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu*

[in] 分布参数 (自由度数)。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

© 2000-2025, MetaQuotes Ltd.

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionT

通过nu参数，计算随机变量x的学生t-分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量x的学生t-分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu参数，计算随机变量数组x[]的学生t-分布函数值。出错情况下它返回false。R语言的pt()模拟。

```
bool  MathCumulativeDistributionT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过nu参数，计算随机变量数组x[]的学生t-分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*nu*

[in] 分布参数 (自由度数)。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileT

对于指定*probability*，函数通过nu参数计算逆学生t-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileT(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数 (自由度数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过nu参数计算逆学生t-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileT(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数 (自由度数)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]* 数组，函数通过nu参数计算逆学生t-分布函数值。出错情况下它返回false。R语言的qt()模拟。

```
double  MathQuantileT(
   const double& probability[],   // 随机变量概率值数组
   const double  nu,              // 分布参数 (自由度数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]* 数组，函数通过nu参数计算逆学生t-分布函数值。出错情况下它返回false。

```
bool  MathQuantileT(
   const double& probability[],   // 随机变量概率值数组
   const double  nu,              // 分布参数 (自由度数)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*nu*

[in] 分布参数 (自由度数)。

*tail*

© 2000-2025, MetaQuotes Ltd.

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomT

通过nu参数生成根据学生t-分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomT(
   const double  nu,              // 分布参数 (自由度数)
   int&          error_code       // 存储错误代码的变量
   );
```

通过nu参数生成根据学生t-分布定律分布的伪随机变量。出错情况下它返回false。R语言的rt()模拟。

```
bool  MathRandomT(
   const double  nu,              // 分布参数 (自由度数)
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsT

通过nu参数计算学生t-分布的前4力矩的理论数值。

```
double  MathMomentsT(
   const double  nu,              // 分布参数 (自由度数)
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

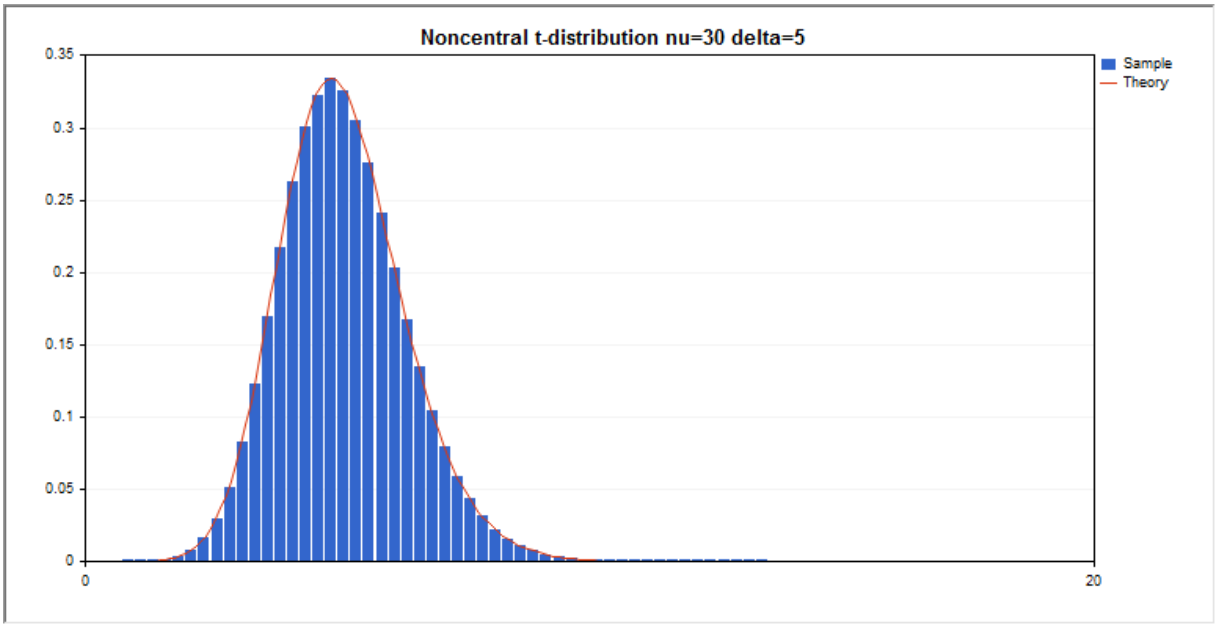
# t-分布

这部分包含了处理非中心学生t-分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据非中心t-分布定律分布的伪随机数。非中心t-分布通过以下公式来定义：

$$f_{NoncentralT}(x|\nu,\delta)=\frac{\nu^{\frac{\nu}{2}}e^{-\frac{\sigma^2}{2}}}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi}\left(\nu+x^2\right)^{\frac{\nu+1}{2}}}\sum_{r=0}^{\infty}\frac{(x\delta)^r}{r!}\left(\frac{2}{\nu+x^2}\right)^{\frac{r}{2}}\Gamma\left(\frac{\nu+r+1}{2}\right)$$

在这里：

- x —随机变量值
- ν — 分布参数（自由度数）
- σ —非中心参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
|---|---|
| MathProbabilityDensityNoncentralT | 计算非中心t-分布的概率密度函数 |
| MathCumulativeDistributionNoncentralT | 计算非中心t-分布函数值 |
| MathQuantileNoncentralT | 计算指定概率逆非中心t-分布函数值 |
| MathRandomNoncentralT | 生成根据非中心学生t-分布定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsNoncentralT | 计算非中心学生t-分布的前4力矩的理论数值 |

© 2000-2025, MetaQuotes Ltd.

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralT.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double nu_par=30;       //自由度数
input double delta_par=5;     // 非中心参数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得非中心学生t-分布的样本
   MathRandomNoncentralT(nu_par,delta_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralT(x2,nu_par,delta_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral t-distribution nu=%G delta=%G",nu_p
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNoncentralT

通过nu和delta参数，计算随机变量x的非中心学生t-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  delta,          // 非中心参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 错误代码变量
   );
```

通过nu和delta参数，计算随机变量x的非中心学生t-分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNoncentralT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数（自由度数）
   const double  delta,          // 非中心参数
   int&          error_code      // 错误代码变量
   );
```

通过nu和delta参数，计算随机变量数组x[]的非中心学生t-分布的概率密度函数值。出错情况下它返回false。R语言的dt()模拟。

```
bool  MathProbabilityDensityNoncentralT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  delta,          // 非中心参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过nu和delta参数，计算随机变量数组x[]的非中心学生t-分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityNoncentralT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数（自由度数）
   const double  delta,          // 非中心参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*nu*

[in] 分布参数（自由度数）。

*delta*

[in] 非中心参数。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNoncentralT

通过nu和delta参数，计算随机变量x的非中心学生t-分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const double  delta,          // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和delta参数，计算随机变量x的非中心学生t-分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNoncentralT(
   const double  x,              // 随机变量值
   const double  nu,             // 分布参数 (自由度数)
   const double  delta,          // 非中心参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过nu和delta参数，计算随机变量数组x[]的非中心学生t-分布的概率分布函数。出错情况下它返回false。R语言的pt()模拟。

```
bool  MathCumulativeDistributionNoncentralT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const double  delta,          // 非中心参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过nu和delta参数，计算随机变量数组x[]的非中心学生t-分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionNoncentralT(
   const double& x[],            // 随机变量值数组
   const double  nu,             // 分布参数 (自由度数)
   const double  delta,          // 非中心参数
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值数组。

*nu*

[in] 分布参数（自由度数）。

*delta*

[in] 非中心参数。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileNoncentralT

对于指定*probability*，函数通过nu和delta参数计算逆非中心学生t-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralT(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   const bool    tail,            // 计算标识，如果lower_tail=false，则计算执行1.0-概率
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过nu和delta参数计算逆非中心学生t-分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNoncentralT(
   const double  probability,     // 随机变量出现的概率值
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过nu和delta参数计算逆非中心学生t-分布函数值。出错情况下它返回false。R语言的qt()模拟。

```
double  MathQuantileNoncentralT(
   const double& probability[],   // 随机变量概率值数组
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   const bool    tail,            // 计算标识，如果lower_tail=false，则计算执行1.0-概率
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过nu和delta参数计算逆非中心学生t-分布函数值。出错情况下它返回false。

```
bool  MathQuantileNoncentralT(
   const double& probability[],   // 随机变量概率值数组
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*nu*

[in] 分布参数（自由度数）。

*delta*

[in] 非中心参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNoncentralT

通过nu和delta参数生成根据非中心学生t-分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNoncentralT(
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   int&          error_code       // 存储错误代码的变量
   );
```

通过nu和delta参数生成根据非中心学生t-分布定律分布的伪随机变量。出错情况下它返回false。R语言的rt()模拟。

```
bool  MathRandomNoncentralT(
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*delta*

[in] 非中心参数。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNoncentralT

通过nu和delta参数计算非中心学生t-分布的前4力矩的理论数值。

```
double  MathMomentsNoncentralT(
   const double  nu,              // 分布参数 (自由度数)
   const double  delta,           // 非中心参数
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*nu*

[in] 分布参数 (自由度数)。

*delta*

[in] 非中心参数。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

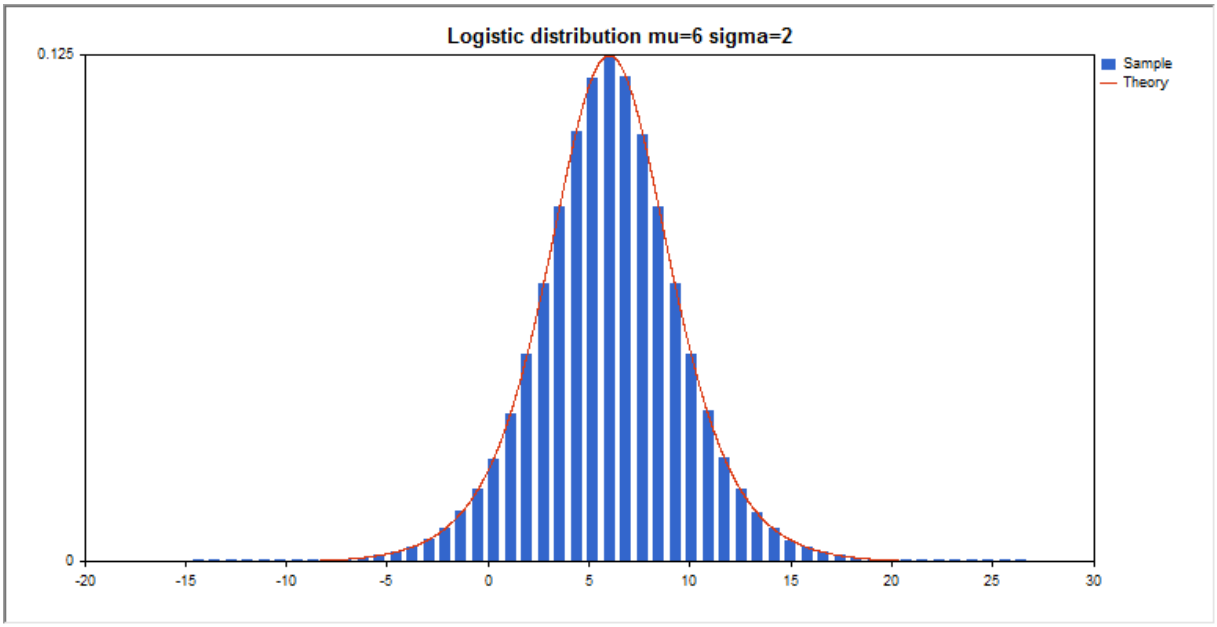
# 逻辑斯谛分布

这部分包含了处理逻辑斯谛分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据逻辑斯谛定律分布的伪随机数。逻辑斯谛分布通过以下公式来定义：

$$f_{Logistic}(x \mid \mu, \sigma) = \frac{e^{\frac{x-\mu}{\sigma}}}{\sigma\left(1+e^{\frac{x-\mu}{\sigma}}\right)^2}$$

在这里：

- x —随机变量值
- μ — 分布的平均参数
- σ —分布的比例参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityLogistic | 计算逻辑斯谛分布的概率密度函数 |
| MathCumulativeDistributionLogistic | 计算逻辑斯谛概率分布函数值 |
| MathQuantileLogistic | 计算指定概率逆逻辑斯谛分布函数值 |
| MathRandomLogistic | 生成根据逻辑斯谛定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsLogistic | 计算逻辑斯谛分布的前4力矩的理论数值 |

© 2000-2025, MetaQuotes Ltd.

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Logistic.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double mu_par=6;          // 分布的平均参数
input double sigma_par=2;       //分布的比例参数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得逻辑斯蒂分布的样本
   MathRandomLogistic(mu_par,sigma_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityLogistic(x2,mu_par,sigma_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

© 2000-2025, MetaQuotes Ltd.

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Logistic distribution mu=%G sigma=%G",mu_par,s
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放Y轴
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(theor_max);
   graphic.YAxis().Min(0);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityLogistic

通过mu和sigma参数，计算随机变量x的逻辑斯谛分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityLogistic(
   const double  x,              // 随机变量值
   const double  mu,             //分布的平均参数
   const double  sigma,          // 分布的比例参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的逻辑斯谛分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityLogistic(
   const double  x,              // 随机变量值
   const double  mu,             //分布的平均参数
   const double  sigma,          // 分布的比例参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x□的逻辑斯谛分布的概率密度函数值。出错情况下它返回false。R语言的dlogis()模拟。

```
bool  MathProbabilityDensityLogistic(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布的平均参数
   const double  sigma,          // 分布的比例参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x□的逻辑斯谛分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityLogistic(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布的平均参数
   const double  sigma,          // 分布的比例参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 分布的平均参数。

*sigma*

[in] 分布的比例参数。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionLogistic

通过mu和sigma参数，计算随机变量x的概率逻辑斯谛分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionLogistic(
   const double  x,              // 随机变量值
   const double  mu,             //分布的平均参数
   const double  sigma,          // 分布的比例参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量x的概率逻辑斯谛分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionLogistic(
   const double  x,              // 随机变量值
   const double  mu,             //分布的平均参数
   const double  sigma,          // 分布的比例参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过mu和sigma参数，计算随机变量数组x[]的概率逻辑斯谛分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionLogistic(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布的平均参数
   const double  sigma,          // 分布的比例参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过mu和sigma参数，计算随机变量数组x[]的概率逻辑斯谛分布函数。出错情况下它返回false。R语言的plogis()模拟。

```
bool  MathCumulativeDistributionLogistic(
   const double& x[],            // 随机变量值数组
   const double  mu,             // 分布的平均参数
   const double  sigma,          // 分布的比例参数
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*mu*

[in] 分布的平均参数。

*sigma*

[in] 分布的比例参数。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileLogistic

对于指定*probability*，函数通过mu和sigma参数计算逆逻辑斯谛分布函数值。出错情况下它返回NaN。

```
double  MathQuantileLogistic(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过mu和sigma参数计算逆逻辑斯谛分布函数值。出错情况下它返回NaN。

```
double  MathQuantileLogistic(
   const double  probability,     // 随机变量出现的概率值
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过mu和sigma参数计算逆逻辑斯谛分布函数值。出错情况下它返回false。R语言的qlogis()模拟。

```
double  MathQuantileLogistic(
   const double& probability[],   // 随机变量概率值数组
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过mu和sigma参数计算逆逻辑斯谛分布函数值。出错情况下它返回false。

```
bool  MathQuantileLogistic(
   const double& probability[],   // 随机变量概率值数组
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*mu*

[in] 分布的平均参数。

*sigma*

[in] 分布的比例参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomLogistic

通过mu和sigma参数生成根据逻辑斯谛分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomLogistic(
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   int&          error_code       // 存储错误代码的变量
   );
```

通过mu和sigma参数生成根据逻辑斯谛分布定律分布的伪随机变量。出错情况下它返回false。R语言的rlogis()模拟。

```
bool  MathRandomLogistic(
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*mu*

[in] 分布的平均参数。

*sigma*

[in] 分布的比例参数。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsLogistic

通过mu和sigma参数计算逻辑斯谛分布的前4力矩的理论数值。

```
double  MathMomentsLogistic(
   const double  mu,              // 分布的平均参数
   const double  sigma,           // 分布的比例参数
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*mu*

[in] 分布的平均参数。

*sigma*

[in] 分布的比例参数。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

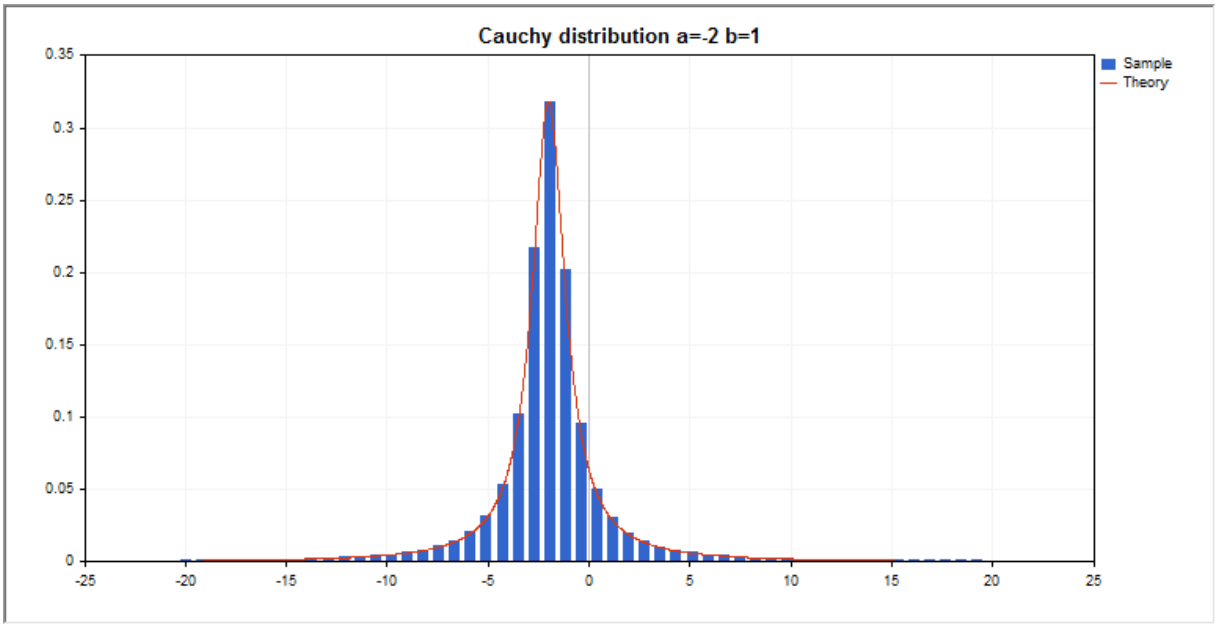
# 柯西分布

这部分包含了处理柯西分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据柯西定律分布的伪随机数。柯西分布通过以下公式来定义：

$$f_{Cauchy}(x|a,b)=\frac{1}{\pi}\frac{b}{\left(b^2+(x-a)^2\right)}$$

在这里：

- x —随机变量值
- a —分布的平均参数
- b —分布的比例参数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityCauchy | 计算柯西分布的概率密度函数 |
| MathCumulativeDistributionCauchy | 计算柯西概率分布函数值 |
| MathQuantileCauchy | 计算指定概率逆柯西分布函数值 |
| MathRandomCauchy | 生成根据柯西定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsCauchy | 计算柯西分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Cauchy.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double a_par=-2;       // 分布的平均参数
input double b_par=1;        // 分布的比例参数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;        // 样本值的数量
   int ncells=51;        // 直方图区间数
   double x[];           // 直方图的区间中心
   double y[];           // 区间内样本数量值下降
   double data[];        // 随机值样本
   double max,min;       // 样本的最大最小值
//--- 获得柯西分布的样本
   MathRandomCauchy(a_par,b_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityCauchy(x2,a_par,b_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Cauchy distribution a=%G b=%G",a_par,b_par));
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1)
      return(false);
   int size=ArraySize(data);
   if(size<cells*10)
      return(false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   Print("min=",minv," max=",maxv);
   minv=-20;
   maxv=20;
   double range=maxv-minv;
   double width=range/cells;
   if(width==0)
      return(false);
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=(int)MathRound((data[i]-minv)/width);
      if(ind>=0 && ind<cells)
         frequency[ind]++;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityCauchy

通过a和b参数，计算随机变量x的柯西分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityCauchy(
   const double  x,              // 随机变量值
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的柯西分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityCauchy(
   const double  x,              // 随机变量值
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的柯西分布的概率密度函数值。出错情况下它返回false。R语言的dcauchy()模拟。

```
bool  MathProbabilityDensityCauchy(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的柯西分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityCauchy(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] 分布的平均参数。

© 2000-2025, MetaQuotes Ltd.

*b*

[in] 分布的比例参数。

*log_mode*

[in] 计算数值对数的标识。如果log_m ode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionCauchy

通过a和b参数，计算随机变量x的柯西分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionCauchy(
   const double  x,              // 随机变量值
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的柯西分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionCauchy(
   const double  x,              // 随机变量值
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的柯西分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionCauchy(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的柯西分布的概率分布函数。出错情况下它返回false。R语言的plogis()模拟。

```
bool  MathCumulativeDistributionCauchy(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

© 2000-2025, MetaQuotes Ltd.

[in] 分布的平均参数。

*b*

[in] 分布的比例参数。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileCauchy

对于指定*probability*，函数通过a和b参数计算逆柯西分布函数值。出错情况下它返回NaN。

```
double  MathQuantileCauchy(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布的平均参数
   const double  b,               // 分布的比例参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过a和b参数计算逆柯西分布函数值。出错情况下它返回NaN。

```
double  MathQuantileCauchy(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布的平均参数
   const double  b,               // 分布的比例参数
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆柯西分布函数值。出错情况下它返回false。R语言的qcauschy()模拟。

```
double  MathQuantileCauchy(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布的平均参数
   const double  b,               // 分布的比例参数
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆柯西分布函数值。出错情况下它返回false。

```
bool  MathQuantileCauchy(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布的平均参数
   const double  b,               // 分布的比例参数
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*a*

[in] 分布的平均参数。

*b*

[in] 分布的比例参数。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomCauchy

通过a和b参数生成根据柯西分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomCauchy(
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数生成根据柯西分布定律分布的伪随机变量。出错情况下它返回false。R语言的rcauchy()模拟。

```
bool  MathRandomCauchy(
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*a*

[in] 分布的平均参数。

*b*

[in] 分布的比例参数。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsCauchy

通过a和b参数计算柯西分布的前4力矩的理论数值。

```
double  MathMomentsCauchy(
   const double  a,              // 分布的平均参数
   const double  b,              // 分布的比例参数
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] 分布的平均参数。

*b*

[in] 分布的比例参数。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

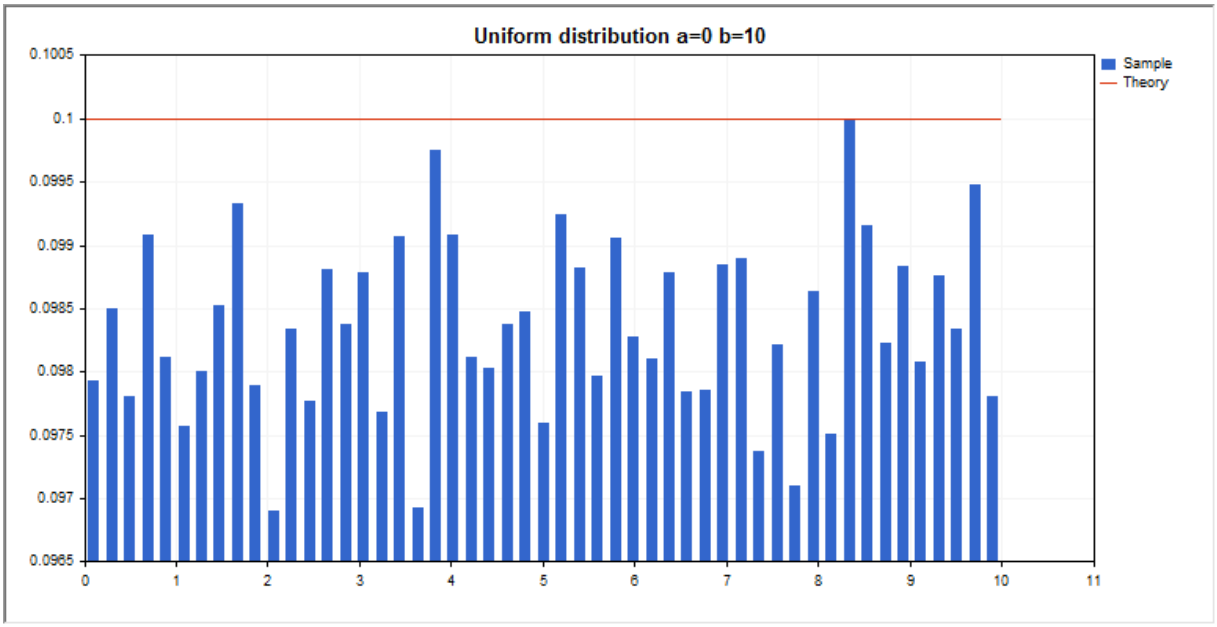
# 均匀分布

这部分包含了处理均匀分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据均匀定律分布的伪随机数。均匀分布通过以下公式来定义：

$$f_{Uniform}(x|a,b)=\frac{1}{b-a}$$

在这里：

- x —随机变量值
- a — 分布参数（下限）
- b —分布参数（上限）




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
|---|---|
| MathProbabilityDensityUniform | 计算均匀分布的概率密度函数 |
| MathCumulativeDistributionUniform | 计算均匀概率分布函数值 |
| MathQuantileUniform | 计算指定概率逆均匀分布函数值 |
| MathRandomUniform | 生成根据均匀定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsUniform | 计算均匀分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\Uniform.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double a_par=0;       // 分布参数a（下限）
input double b_par=10;      // 分布参数b（上限）
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=51;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得均匀分布的样本
   MathRandomUniform(a_par,b_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityUniform(x2,a_par,b_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.BackgroundMain(StringFormat("Uniform distribution a=%G b=%G",a_par,b_par));
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityUniform

通过a和b参数，计算随机变量x的均匀分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityUniform(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的均匀分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityUniform(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的均匀分布的概率密度函数值。出错情况下它返回false。R语言的dunif()模拟。

```
bool  MathProbabilityDensityUniform(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的均匀分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityUniform(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] 分布参数a（下限）。

© 2000-2025, MetaQuotes Ltd.

*b*

[in] 分布参数b（上限）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionUniform

通过a和b参数，计算随机变量x的均匀分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionUniform(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的均匀分布的概率分布函数。出错情况下它返回NaN。

```
double  MathCumulativeDistributionUniform(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的均匀分布的概率分布函数。出错情况下它返回false。

```
bool  MathCumulativeDistributionUniform(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的均匀分布的概率分布函数。出错情况下它返回false。R语言的punif()模拟。

```
bool  MathCumulativeDistributionUniform(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

© 2000-2025, MetaQuotes Ltd.

[in] 分布参数a（下限）。

*b*

[in] 分布参数b（上限）。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileUniform

对于指定*probability*，函数通过a和b参数计算逆均匀分布函数值。出错情况下它返回NaN。

```
double  MathQuantileUniform(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布参数a（下限）
   const double  b,               // 分布参数b（上限）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过a和b参数计算逆均匀分布函数值。出错情况下它返回NaN。

```
double  MathQuantileUniform(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布参数a（下限）
   const double  b,               // 分布参数b（上限）
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆均匀分布函数值。出错情况下它返回false。R语言的qcauschy()模拟。

```
double  MathQuantileUniform(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布参数a（下限）
   const double  b,               // 分布参数b（上限）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆均匀分布函数值。出错情况下它返回false。

```
bool  MathQuantileUniform(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布参数a（下限）
   const double  b,               // 分布参数b（上限）
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*a*

[in] 分布参数a（下限）。

*b*

[in] 分布参数b（上限）。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomUniform

通过a和b参数生成根据均匀分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomUniform(
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数生成根据均匀分布定律分布的伪随机变量。出错情况下它返回false。R语言的runif()模拟。

```
bool  MathRandomUniform(
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*a*

[in] 分布参数a（下限）。

*b*

[in] 分布参数b（上限）。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsUniform

通过a和b参数计算均匀分布的前4力矩的理论数值。

```
double  MathMomentsUniform(
   const double  a,              // 分布参数a（下限）
   const double  b,              // 分布参数b（上限）
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] 分布参数a（下限）。

*b*

[in] 分布参数b（上限）。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

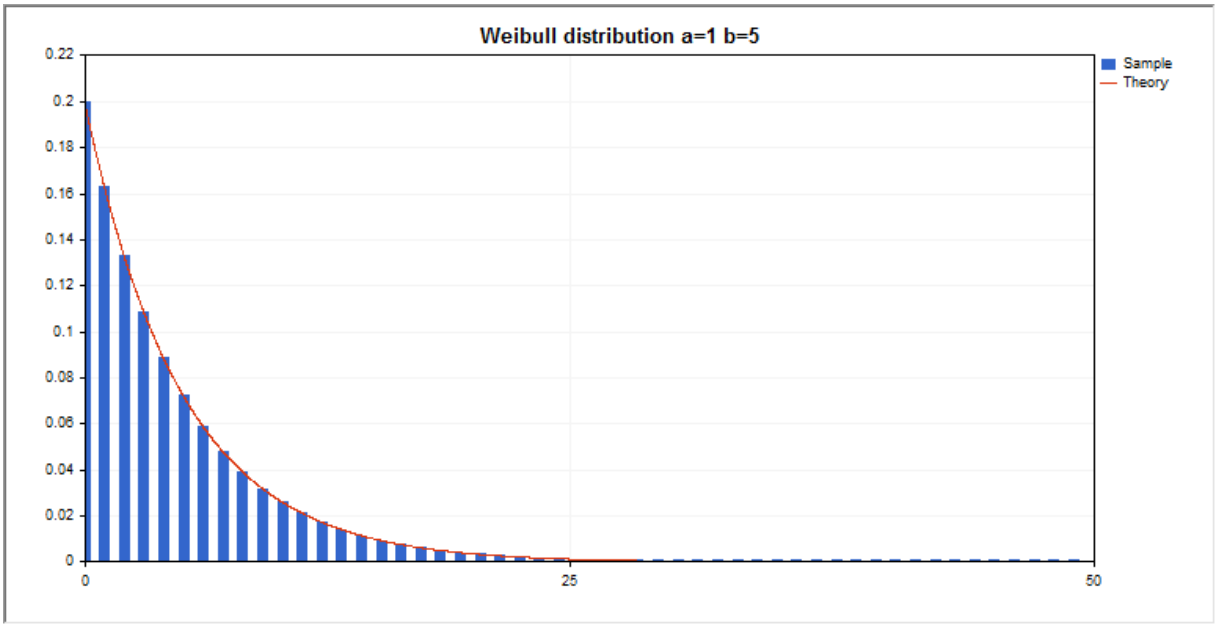
# 威布尔分布

这部分包含了处理威布尔分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据威布尔定律分布的伪随机数。威布尔分布通过以下公式定义：

$$f_{Weibull}(x|a,b)=\frac{a}{b}\left(\frac{x}{b}\right)^{a-1}e^{-\left(\frac{x}{b}\right)^{a}}$$

在这里：

- x —随机变量值
- a — 分布参数(shape)
- b —分布参数(scale)




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityWeibull | 计算威布尔分布的概率密度函数 |
| MathCumulativeDistributionWeibull | 计算威布尔概率分布函数值 |
| MathQuantileWeibull | 计算指定概率逆威布尔分布函数值 |
| MathRandomWeibull | 生成根据威布尔定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsWeibull | 计算威布尔分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Weibull.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double a_par=1;       // 分布参数 (shape)
input double b_par=5;       // 分布参数 (scale)
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;         // 样本值的数量
   int ncells=51;         // 直方图区间数
   double x[];            // 直方图的区间中心
   double y[];            // 区间内样本数量值下降
   double data[];         // 随机值样本
   double max,min;        // 样本的最大最小值
//--- 获得威布尔分布的样本
   MathRandomWeibull(a_par,b_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityWeibull(x2,a_par,b_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Weibull distribution a=%G b=%G",a_par,b_par));
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放X轴
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(max);
   graphic.XAxis().Min(min);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityWeibull

通过a和b参数，计算随机变量x的威布尔分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityWeibull(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的威布尔分布的概率密度函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityWeibull(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的威布尔分布的概率密度函数值。出错情况下它返回false。R语言的dweibull()模拟。

```
bool  MathProbabilityDensityWeibull(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的威布尔分布的概率密度函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityWeibull(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

[in] 分布参数(scale)。

© 2000-2025, MetaQuotes Ltd.

*b*

[in] 分布参数(shape)。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionWeibull

通过a和b参数，计算随机变量x的威布尔分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionWeibull(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量x的威布尔分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionWeibull(
   const double  x,              // 随机变量值
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数，计算随机变量数组x[]的威布尔分布函数值。出错情况下它返回false。R语言的pweibull()模拟。

```
bool  MathCumulativeDistributionWeibull(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过a和b参数，计算随机变量数组x[]的威布尔分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionWeibull(
   const double& x[],            // 随机变量值数组
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*a*

© 2000-2025, MetaQuotes Ltd.

[in] 分布参数(scale)。

*b*

[in] 分布参数(shape)。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileWeibull

对于指定*probability*，函数通过a和b参数计算逆威布尔分布函数值。出错情况下它返回NaN。

```
double  MathQuantileWeibull(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布参数(shape)
   const double  b,               // 分布参数(scale)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过a和b参数计算逆威布尔分布函数值。出错情况下它返回NaN。

```
double  MathQuantileWeibull(
   const double  probability,     // 随机变量出现的概率值
   const double  a,               // 分布参数(shape)
   const double  b,               // 分布参数(scale)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆威布尔分布函数值。出错情况下它返回false。R语言的qweibull()模拟。

```
double  MathQuantileWeibull(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布参数(shape)
   const double  b,               // 分布参数(scale)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过a和b参数计算逆威布尔分布函数值。出错情况下它返回false。

```
bool  MathQuantileWeibull(
   const double& probability[],   // 随机变量概率值数组
   const double  a,               // 分布参数(shape)
   const double  b,               // 分布参数(scale)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

© 2000-2025, MetaQuotes Ltd.

*a*

[in] 分布参数(scale)。

*b*

[in] 分布参数(shape)。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomWeibull

通过a和b参数生成根据威布尔分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomWeibull(
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   int&          error_code      // 存储错误代码的变量
   );
```

通过a和b参数生成根据威布尔分布定律分布的伪随机变量。出错情况下它返回false。R语言的rweibull()模拟。

```
bool  MathRandomWeibull(
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*a*

[in] 分布参数(scale)。

*b*

[in] 分布参数(shape)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out]获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsWeibull

通过a和b参数计算威布尔分布的前4力矩的理论数值。

```
double  MathMomentsWeibull(
   const double  a,              // 分布参数(shape)
   const double  b,              // 分布参数(scale)
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*a*

[in] 分布参数(scale)。

*b*

[in] 分布参数(shape)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

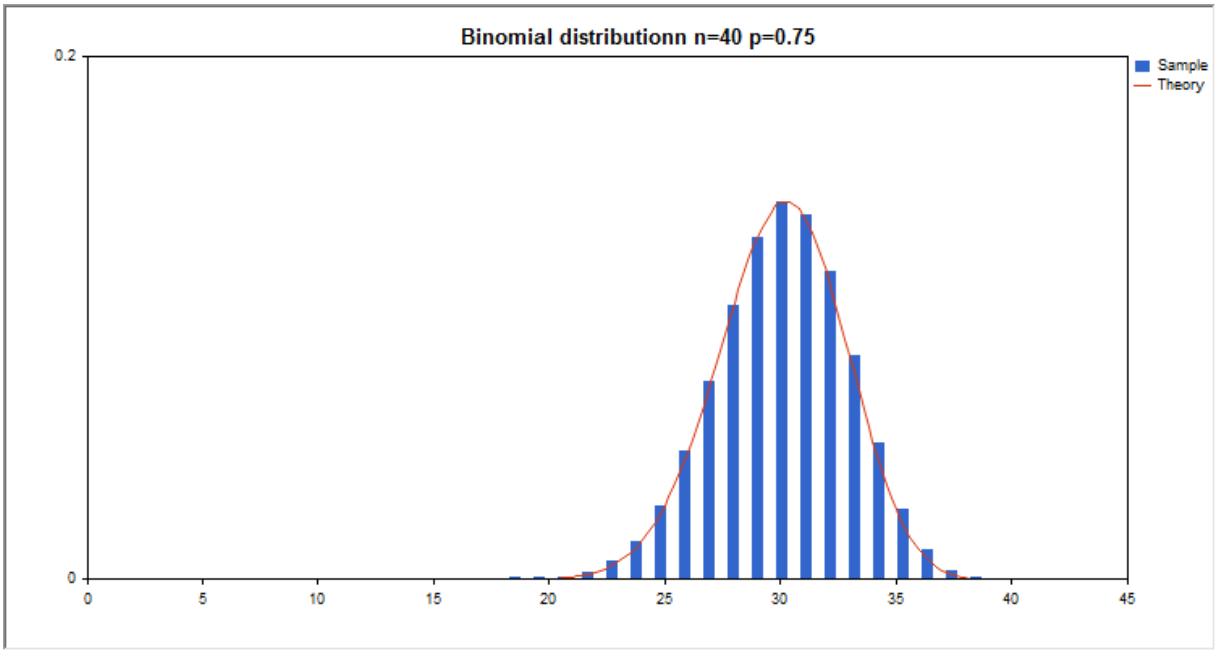
# 二项式分布

这部分包含了处理二项式分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据二项式定律分布的伪随机数。二项式分布通过以下公式来定义：

$$f_{Binomial}(x|n,p)=\binom{n}{x}p^x(1-p)^{n-x}$$

在这里：

- x —随机变量值
- n —测试数
- p —每次测试的成功概率




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityBinomial | 计算二项式分布的概率密度函数 |
| MathCumulativeDistributionBinomial | 计算二项式概率分布函数值 |
| MathQuantileBinomial | 计算指定概率逆二项式分布函数值 |
| MathRandomBinomial | 生成根据二项式定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsBinomial | 计算二项式分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Binomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double n_par=40;         // 测试数
input double p_par=0.75;       // 每次测试的成功概率
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=20;          // 直方图的区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得二项式分布的样本
   MathRandomBinomial(n_par,p_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityBinomial(x2,n_par,p_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Binomial distributionn n=%G p=%G",n_par,p_par)
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityBinomial

通过n和p参数，计算随机变量x的二项式分布的概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityBinomial(
   const double  x,              // 随机变量值
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过n和p参数，计算随机变量x的二项式分布的概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityBinomial(
   const double  x,              // 随机变量值
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过n和p参数，计算随机变量数组x[]的二项式分布的概率质量函数值。出错情况下它返回false。R语言的dbinom()模拟。

```
bool  MathProbabilityDensityBinomial(
   const double& x[],            // 随机变量值数组
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过n和p参数，计算随机变量数组x[]的二项式分布的概率质量函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityBinomial(
   const double& x[],            // 随机变量值数组
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*n*

[in] 分布参数  测试数）。

© 2000-2025, MetaQuotes Ltd.

*p*

[in] 分布参数（每次测试中事件发生的概率）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionBinomial

通过n和p参数，计算随机变量x的二项式定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionBinomial(
   const double  x,              // 随机变量值
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过n和p参数，计算随机变量x的二项式定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionBinomial(
   const double  x,              // 随机变量值
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过n和p参数，计算随机变量数组x[]的二项式定律下概率分布函数值。出错情况下它返回false。R语言的pweibull()模拟。

```
bool  MathCumulativeDistributionBinomial(
   const double& x[],            // 随机变量值数组
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过随机变量x[]数组的n和p参数计算二项式定律下概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionBinomial(
   const double& x[],            // 随机变量值数组
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*n*

© 2000-2025, MetaQuotes Ltd.

[in] 分布参数（测试数）。

*p*

[in] 分布参数（每次测试中事件发生的概率）。

*tail*

[in] 计算标识。如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileBinomial

对于指定*probability*，函数通过n和p参数计算二项式定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileBinomial(
   const double  probability,     // 随机变量出现的概率值
   const double  n,               // 分布参数  测试数）
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过n和p参数计算二项式定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileBinomial(
   const double  probability,     // 随机变量出现的概率值
   const double  n,               // 分布参数  测试数）
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过n和p参数计算二项式定律下逆分布函数值。出错情况下它返回false。R语言的qbinom()模拟。

```
double  MathQuantileBinomial(
   const double& probability[],   // 随机变量概率值数组
   const double  n,               // 分布参数  测试数）
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过n和p参数计算二项式定律下逆分布函数值。出错情况下它返回false。

```
bool  MathQuantileBinomial(
   const double& probability[],   // 随机变量概率值数组
   const double  n,               // 分布参数  测试数）
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*n*

[in] 分布参数（测试数）。

*p*

[in] 分布参数（每次测试中事件发生的概率）。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomBinomial

通过n和p参数生成根据负二项式分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomBinomial(
   const double  n,              // 分布参数（测试数）
   const double  p,              // 分布参数（每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过n和p参数生成根据二项式分布定律分布的伪随机变量。出错情况下它返回false。R语言的rweibull()模拟。

```
bool  MathRandomBinomial(
   const double  n,              // 分布参数（测试数）
   const double  p,              // 分布参数（每次测试中事件发生的概率）
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*n*

[in] 分布参数（测试数）。

*p*

[in] 分布参数（每次测试中事件发生的概率）。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsBinomial

通过n和p参数计算二项式分布的前4力矩的理论数值。

```
double  MathMomentsBinomial(
   const double  n,              // 分布参数  测试数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*n*

[in] 分布参数  测试数）。

*p*

[in] 分布参数  每次测试中事件发生的概率）。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

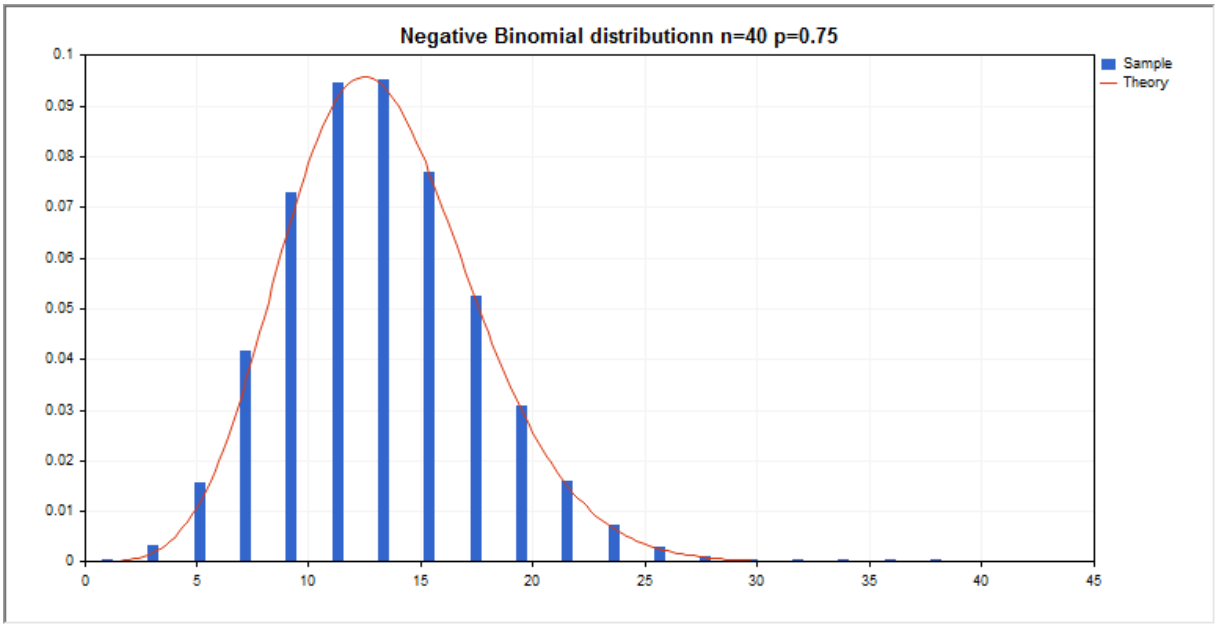
# 负二项式分布

这部分包含了处理负二项式分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据负二项式定律分布的伪随机数。负二项式分布通过以下公式来定义：

$$f_{NegativeBinomial}(x|r,p)=\frac{\Gamma(r+x)}{\Gamma(r)\Gamma(x+1)}p^r(1-p)^x$$

在这里：

- x —随机变量值
- r —成功测试的数量
- p —成功的概率




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityNegativeBinomial | 计算负二项式分布的概率密度函数 |
| MathCumulativeDistributionNegativeBinomial | 计算负二项式概率分布函数值 |
| MathQuantileNegativeBinomial | 计算指定概率逆负二项式分布函数值 |
| MathRandomNegativeBinomial | 生成根据负二项式定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsNegativeBinomial | 计算负二项式分布的前4力矩的理论数值 |

示例：

© 2000-2025, MetaQuotes Ltd.

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NegativeBinomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double n_par=40;         // 测试数
input double p_par=0.75;       // 每次测试的成功概率
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;         // 样本值的数量
   int ncells=19;         // 直方图区间数
   double x[];            // 直方图的区间中心
   double y[];            // 区间内样本数量值下降
   double data[];         // 随机值样本
   double max,min;        // 样本的最大最小值
//--- 获得负二项式分布的样本
   MathRandomNegativeBinomial(n_par,p_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityNegativeBinomial(x2,n_par,p_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Negative Binomial distributionn n=%G p=%G",n_p
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityNegativeBinomial

通过r和p参数，计算随机变量x的负二项式分布的概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNegativeBinomial(
   const double  x,              // 随机变量值 整数）
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过r和p参数，计算随机变量x的负二项式分布的概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityNegativeBinomial(
   const double  x,              // 随机变量值 整数）
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   int&          error_code      // 存储错误代码的变量
   );
```

通过r和p参数，计算随机变量数组x[]的负二项式分布的概率质量函数值。出错情况下它返回false。R语言的dnbinom()模拟。

```
bool  MathProbabilityDensityNegativeBinomial(
   const double& x[],            // 随机变量值数组
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过r和p参数，计算随机变量数组x[]的负二项式分布的概率质量函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityNegativeBinomial(
   const double& x[],            // 随机变量值数组
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*r*

[in] 成功测试的数量

© 2000-2025, MetaQuotes Ltd.

*p*

[in] 成功的概率。

*log_mode*

[in] 计算数值对数的标识。如果log_m ode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionNegativeBinomial

通过r和p参数，计算随机变量x的负二项式定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNegativeBinomial(
   const double  x,              // 随机变量值 整数）
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过r和p参数，计算随机变量x的负二项式定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionNegativeBinomial(
   const double  x,              // 随机变量值 整数）
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   int&          error_code      // 存储错误代码的变量
   );
```

通过r和p参数，计算随机变量数组x[]的负二项式定律下概率分布函数值。出错情况下它返回false。R语言的pweibull()模拟。

```
bool  MathCumulativeDistributionNegativeBinomial(
   const double& x[],            // 随机变量值数组
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过r和p参数，计算随机变量数组x[]的负二项式定律下概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionNegativeBinomial(
   const double& x[],            // 随机变量值数组
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

© 2000-2025, MetaQuotes Ltd.

*r*

[in] 成功测试的数量。

*p*

[in] 成功的概率。

*tail*

[in] 计算标识，如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileNegativeBinomial

对于指定*probability*，函数通过r和p参数计算负二项式定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNegativeBinomial(
   const double  probability,     // 随机变量出现的概率值
   const double  r,               // 成功测试的数量
   const double  p,               // 成功的概率
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过r和p参数计算负二项式定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileNegativeBinomial(
   const double  probability,     // 随机变量出现的概率值
   const double  r,               // 成功测试的数量
   const double  p,               // 成功的概率
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过r和p参数计算负二项式定律下逆分布函数值。出错情况下它返回false。R语言的qnbinom()模拟。

```
double  MathQuantileNegativeBinomial(
   const double& probability[],   // 随机变量概率值数组
   const double  r,               // 成功测试的数量
   const double  p,               // 成功的概率
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过r和p参数计算负二项式定律下逆分布函数值。出错情况下它返回false。

```
bool  MathQuantileNegativeBinomial(
   const double& probability[],   // 随机变量概率值数组
   const double  r,               // 成功测试的数量
   const double  p,               // 成功的概率
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量概率值数组。

*r*

[in] 成功测试的数量。

*p*

[in] 成功的概率。

*tail*

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomNegativeBinomial

通过r和p参数生成根据负二项式分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomNegativeBinomial(
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   int&          error_code      // 存储错误代码的变量
   );
```

通过r和p参数生成根据负二项式分布定律分布的伪随机变量。出错情况下它返回false。R语言的rweibull()模拟。

```
bool  MathRandomNegativeBinomial(
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*r*

[in] 成功测试的数量。

*p*

[in] 成功的概率。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsNegativeBinomial

通过r和p参数计算负二项式分布的前4力矩的理论数值。

```
double  MathMomentsNegativeBinomial(
   const double  r,              // 成功测试的数量
   const double  p,              // 成功的概率
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*r*

[in] 成功测试的数量。

*p*

[in] 成功的概率。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

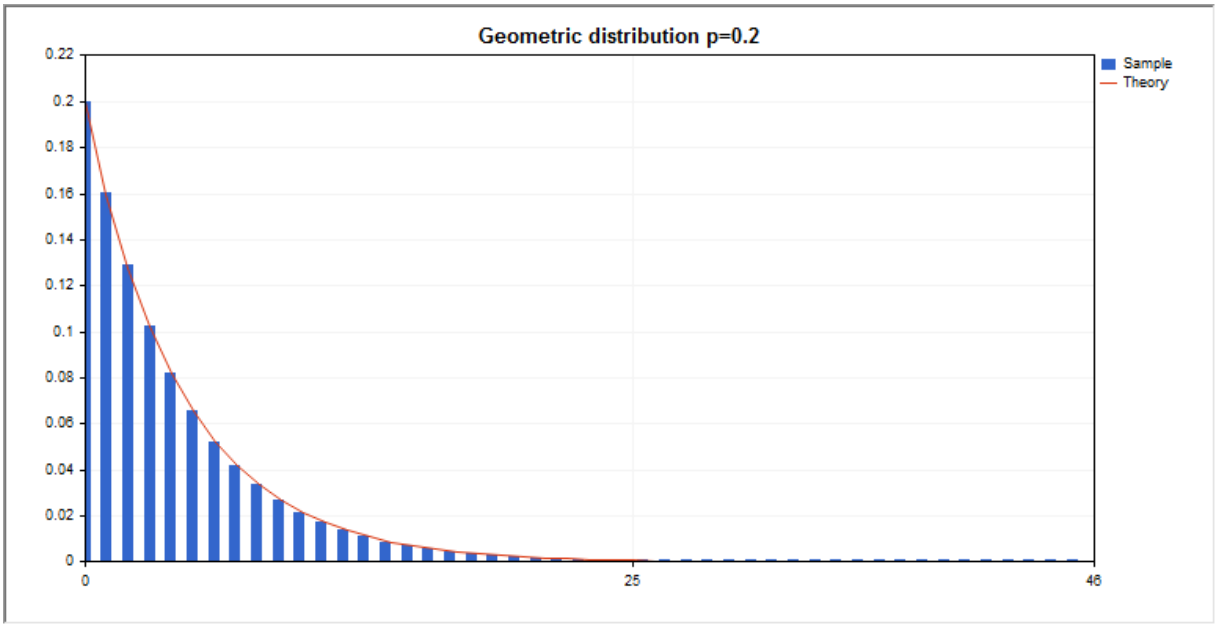
# 几何分布

这部分包含了处理几何分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据几何定律分布的伪随机数。几何分布通过以下公式来定义：

$$f_{Geometric}(x|p) = p(1-p)^x$$

在这里：

- x —随机变量值（整数）
- p —每次测试中事件发生的概率




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityGeometric | 计算几何分布的概率密度函数 |
| MathCumulativeDistributionGeometric | 计算几何概率分布函数值 |
| MathQuantileGeometric | 计算指定概率逆几何分布函数值 |
| MathRandomGeometric | 生成根据几何定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsGeometric | 计算几何分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Geometric.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double p_par=0.2;       // 每次测试中事件发生的概率
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=47;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得几何分布的样本
   MathRandomGeometric(p_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(0,ncells,1,x2);
   MathProbabilityDensityGeometric(x2,p_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Geometric distribution p=%G",p_par));
   graphic.BackgroundMainSize(16);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 禁止自动缩放X轴
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(max);
   graphic.XAxis().Min(min);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
```

© 2000-2025, MetaQuotes Ltd.

```
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityGeometric

通过p参数，计算随机变量x的几何分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityGeometric(
   const double  x,              // 随机变量值 整数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过p参数，计算随机变量x的几何分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityGeometric(
   const double  x,              // 随机变量值 整数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过p参数，计算随机变量数组x[]的几何分布下概率质量函数值。出错情况下它返回false。R语言的dgeom()模拟。

```
bool  MathProbabilityDensityGeometric(
   const double& x[],            // 随机变量值数组
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过p参数，计算随机变量数组x[]的几何分布下概率质量函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityGeometric(
   const double& x[],            // 随机变量值数组
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*p*

[in] 分布参数  每次测试中事件发生的概率）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

© 2000-2025, MetaQuotes Ltd.

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionGeometric

通过p参数，计算随机变量x的几何定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionGeometric(
   const double  x,              // 随机变量值 整数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过p参数，计算随机变量x的几何定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionGeometric(
   const double  x,              // 随机变量值 整数）
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过p参数，计算随机变量数组x[]的几何定律下概率分布函数值。出错情况下它返回false。
Analog of the pgeom() in R.

```
bool  MathCumulativeDistributionGeometric(
   const double& x[],            // 随机变量值数组
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 概率函数值数组
   );
```

通过p参数，计算随机变量数组x[]的几何定律下概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionGeometric(
   const double& x[],            // 随机变量值数组
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   double&       result[]        // 概率函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*p*

[in] 分布参数  每次测试中事件发生的概率）。

*tail*

[in] Flag of calculation, if tail=true, then the probability of random variable not exceeding x is calculated.

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileGeometric

对于指定*probability*，函数通过p参数计算几何定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileGeometric(
   const double  probability,     // 随机变量出现的概率值
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过p参数计算几何定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileGeometric(
   const double  probability,     // 随机变量出现的概率值
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过p参数计算几何定律下逆分布函数值。出错情况下它返回false。R语言的qgeom()模拟。

```
double  MathQuantileGeometric(
   const double& probability[],   // 随机变量概率值数组
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过p参数计算几何定律下逆分布函数值。出错情况下它返回false。

```
bool  MathQuantileGeometric(
   const double& probability[],   // 随机变量概率值数组
   const double  p,               // 分布参数  每次测试中事件发生的概率）
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*p*

[in] 分布参数  每次测试中事件发生的概率）。

*tail*

© 2000-2025, MetaQuotes Ltd.

[in] 计算标识，如果false，则执行1.0-概率的计算。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomGeometric

通过p参数生成根据几何分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomGeometric(
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   int&          error_code      // 存储错误代码的变量
   );
```

通过p参数生成根据几何分布定律分布的伪随机变量。出错情况下它返回false。R语言的rgeom()模拟。

```
bool  MathRandomGeometric(
   const double  p,              // 分布参数  每次测试中事件发生的概率）
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*p*

[in] 分布参数  每次测试中事件发生的概率）。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsGeometric

通过p参数计算几何分布的前4力矩的理论数值。

```
double  MathMomentsGeometric(
   const double  p,              // 分布参数  (每次测试中事件发生的概率)
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*p*

[in] 分布参数  (每次测试中事件发生的概率)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

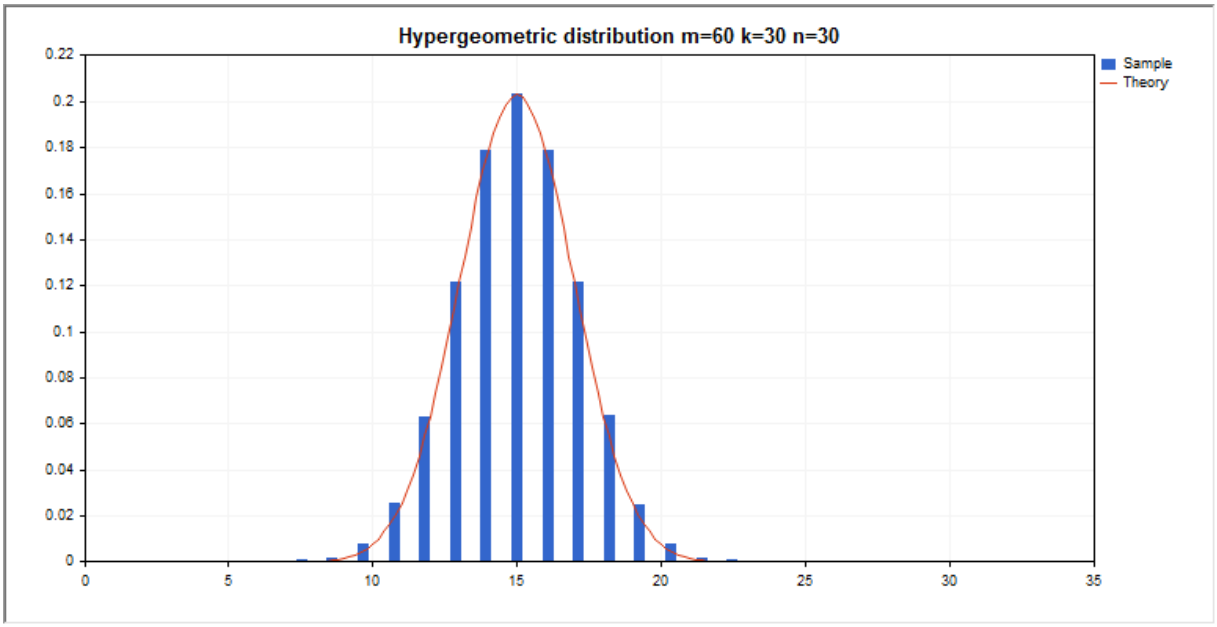
# 超几何分布

这部分包含了处理超几何分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据超几何定律分布的伪随机数。超几何分布通过以下公式来定义：

$$f_{Hypergeometric}(x|m,k,n)=\frac{\binom{k}{x}\binom{m-k}{n-x}}{\binom{m}{n}}$$

在这里：

- x —随机变量值（整数）
- m —对象总数
- k —包含所需特性的对象数
- n —对象绘制数




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityHypergeometric | 计算超几何分布的概率密度函数 |
| MathCumulativeDistributionHypergeometric | 计算超几何概率分布函数值 |
| MathQuantileHypergeometric | 计算指定概率逆超几何分布函数值 |
| MathRandomHypergeometric | 生成根据超几何定律分布的伪随机变量/伪随机变量数组 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathMomentsHypergeometric | 计算超几何分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Hypergeometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- 输入参数
input double m_par=60;       // 对象总数
input double k_par=30;       // 包含所需特性的对象数
input double n_par=30;       // 绘制对象数
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;          // 样本值的数量
   int ncells=15;          // 直方图区间数
   double x[];             // 直方图的区间中心
   double y[];             // 区间内样本数量值下降
   double data[];          // 随机值样本
   double max,min;         // 样本的最大最小值
//--- 获得超几何分布的样本
   MathRandomHypergeometric(m_par,k_par,n_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityHypergeometric(x2,m_par,k_par,n_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
```

© 2000-2025, MetaQuotes Ltd.

```
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Hypergeometric distribution m=%G k=%G n=%G",m_
   graphic.BackgroundMainSize(16);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityHypergeometric

通过m，k和n参数，计算随机变量x的超几何分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityHypergeometric(
   const double  x,              // 随机变量值 整数)
   const double  m,              // 对象总数 整数)
   const double  k,              // 包含所需特性的对象数 整数)
   const double  n,              // 对象绘制数 整数)
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过m，k和n参数，计算随机变量x的超几何分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityHypergeometric(
   const double  x,              // 随机变量值 整数)
   const double  m,              // 对象总数 整数)
   const double  k,              // 包含所需特性的对象数 整数)
   const double  n,              // 对象绘制数 整数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过m，k和n参数，计算随机变量数组x[]的超几何分布下概率质量函数值。出错情况下它返回false。R语言的dhyper()模拟。

```
bool  MathProbabilityDensityHypergeometric(
   const double& x[],            // 随机变量值数组
   const double  m,              //对象总数 整数)
   const double  k,              //包含所需特性的对象数 整数)
   const double  n,              //对象绘制数 整数)
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过m，k和n参数，计算随机变量数组x[]的超几何分布下概率质量函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityHypergeometric(
   const double& x[],            // 随机变量值数组
   const double  m,              //对象总数 整数)
   const double  k,              //包含所需特性的对象数 整数)
   const double  n,              //对象绘制数 整数)
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值数组。

*m*

[in] 对象总数（整数）。

*k*

[in] 包含所需特性的对象数（整数）。

*n*

[in] 对象绘制数（整数）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionHypergeometric

通过m，k和n参数，计算随机变量x的超几何定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionHypergeometric(
   const double  x,              // 随机变量值 整数)
   const double  m,              // 对象总数 整数)
   const double  k,              // 包含所需特性的对象数 整数)
   const double  n,              // 对象绘制数 整数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过m，k和n参数，计算随机变量x的超几何定律下概率分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionHypergeometric(
   const double  x,              // 随机变量值 整数)
   const double  m,              // 对象总数 整数)
   const double  k,              // 包含所需特性的对象数 整数)
   const double  n,              // 对象绘制数 整数)
   int&          error_code      // 存储错误代码的变量
   );
```

通过m，k和n参数，计算随机变量数组x[]的超几何定律下概率分布函数值。出错情况下它返回false。R语言的dhyper()模拟。

```
bool  MathCumulativeDistributionHypergeometric(
   const double& x[],            // 随机变量值数组
   const double  m,              //对象总数 整数)
   const double  k,              //包含所需特性的对象数 整数)
   const double  n,              //对象绘制数 整数)
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 分布函数值数组
   );
```

通过m，k和n参数，计算随机变量数组x[]的超几何定律下概率分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionHypergeometric(
   const double& x[],            // 随机变量值数组
   const double  m,              //对象总数 整数)
   const double  k,              //包含所需特性的对象数 整数)
   const double  n,              //对象绘制数 整数)
   double&       result[]        // 分布函数值数组
   );
```

参数

*x*

© 2000-2025, MetaQuotes Ltd.

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*m*

[in] 对象总数（整数）。

*k*

[in] 包含所需特性的对象数（整数）。

*n*

[in] 对象绘制数（整数）。

*tail*

[in] 计算标识，如果true，则计算不超过随机变量x的概率。

*log_mode*

[in] 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 分布函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantileHypergeometric

对于指定*probability*，函数通过m，k和n参数计算超几何定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileHypergeometric(
   const double  probability,     // 随机变量出现的概率值
   const double  m,               //对象总数 整数)
   const double  k,               //包含所需特性的对象数 整数)
   const double  n,               //对象绘制数 整数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过m，k和n参数计算超几何定律下逆分布函数值。出错情况下它返回NaN。

```
double  MathQuantileHypergeometric(
   const double  probability,     // 随机变量出现的概率值
   const double  m,               //对象总数 整数)
   const double  k,               //包含所需特性的对象数 整数)
   const double  n,               //对象绘制数 整数)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过m，k和n参数计算超几何定律下逆分布函数值。出错情况下它返回false。R语言的qhyper()模拟。

```
double  MathQuantileHypergeometric(
   const double& probability[],   // 随机变量概率值数组
   const double  m,               //对象总数 整数)
   const double  k,               //包含所需特性的对象数 整数)
   const double  n,               //对象绘制数 整数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过m，k和n参数计算超几何定律下逆分布函数值。出错情况下它返回false。

```
bool  MathQuantileHypergeometric(
   const double& probability[],   // 随机变量概率值数组
   const double  m,               //对象总数 整数)
   const double  k,               //包含所需特性的对象数 整数)
   const double  n,               //对象绘制数 整数)
   double&       result[]         // 四分位数值数组
   );
```

© 2000-2025, MetaQuotes Ltd.

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*m*

[in] 对象总数（整数）。

*k*

[in] 包含所需特性的对象数（整数）。

*n*

[in] 对象绘制数（整数）。

*tail*

[in] 计算标识，如果tail=false，则计算执行1.0-概率。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomHypergeometric

通过m，n和k参数生成根据超几何分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomHypergeometric(
   const double  m,              //对象总数 整数）
   const double  k,              //包含所需特性的对象数 整数）
   const double  n,              //对象绘制数 整数）
   int&          error_code      // 存储错误代码的变量
   );
```

通过m，n和k参数生成根据超几何分布定律分布的伪随机变量。出错情况下它返回false。R语言的rgeom()模拟。

```
bool  MathRandomHypergeometric(
   const double  m,              //对象总数 整数）
   const double  k,              //包含所需特性的对象数 整数）
   const double  n,              //对象绘制数 整数）
   const int     data_count,     // 所需的数据量
   double&       result[]        // 伪随机变量值数组
   );
```

参数

*m*

[in] 对象总数 (整数)。

*k*

[in] 包含所需特性的对象数 (整数)。

*n*

[in] 对象绘制数 (整数)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsHypergeometric

通过m，n和k参数计算超几何分布的前4力矩的理论数值。

```
double  MathMomentsHypergeometric(
   const double  m,              //对象总数 整数）
   const double  k,              //包含所需特性的对象数 整数）
   const double  n,              //对象绘制数 整数）
   double&       mean,           // 平均值变量
   double&       variance,       // 方差变量
   double&       skewness,       // 偏度变量
   double&       kurtosis,       // 峰度变量
   int&          error_code      // 错误代码变量
   );
```

参数

*m*

[in] 对象总数 (整数)。

*k*

[in] 包含所需特性的对象数 (整数)。

*n*

[in] 对象绘制数 (整数)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

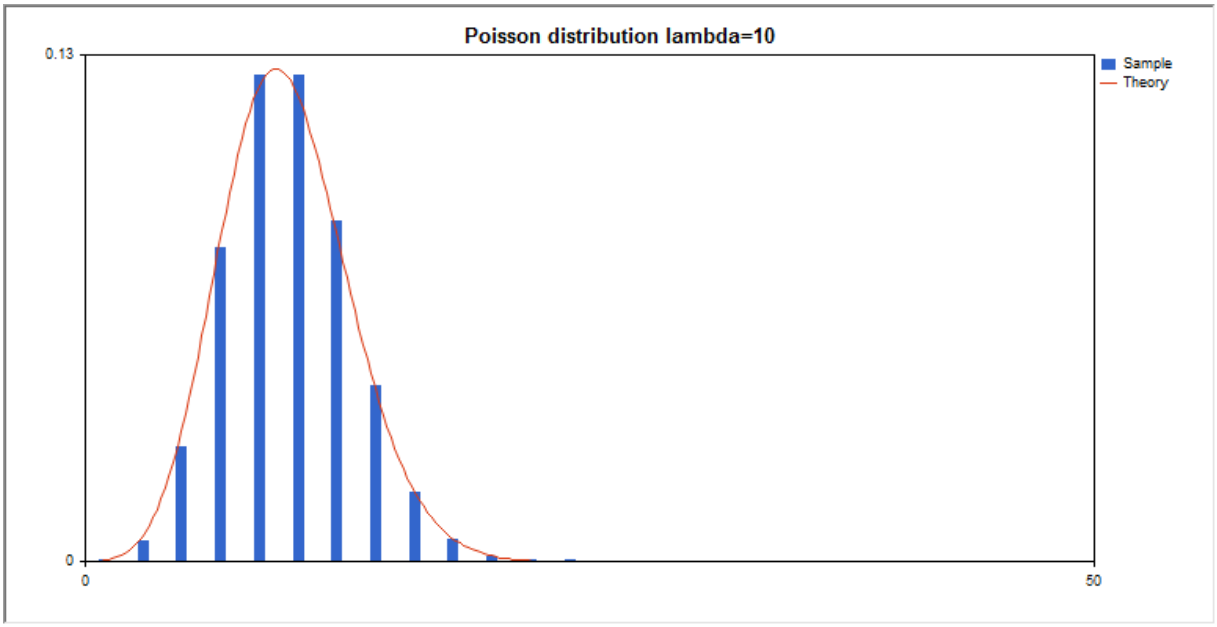
# 泊松分布

这部分包含了处理泊松分布的所有函数。它们可以计算密度，概率，四分位数以及生成根据泊松定律分布的伪随机数。泊松分布通过以下公式来定义：

$$f_{Poisson}(x|\lambda)=\frac{\lambda^x}{x!}e^{-\lambda}$$

在这里：

- x —随机变量值
- λ—分布参数（平均数）




除了计算单独的随机变量以外，程序库还具有处理随机变量数组的能力。

| 函数 | 描述 |
| --- | --- |
| MathProbabilityDensityPoisson | 计算泊松分布的概率密度函数 |
| MathCumulativeDistributionPoisson | 计算泊松概率分布函数值 |
| MathQuantilePoisson | 计算指定概率逆泊松分布函数值 |
| MathRandomPoisson | 生成根据泊松定律分布的伪随机变量/伪随机变量数组 |
| MathMomentsPoisson | 计算泊松分布的前4力矩的理论数值 |

示例：

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Poisson.mqh>
#include <Math\Stat\Math.mqh>
```

© 2000-2025, MetaQuotes Ltd.

```
#property script_show_inputs
//--- 输入参数
input double lambda_par=10;      // 分布参数（平均数）
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 隐藏价格图表
   ChartSetInteger(0,CHART_SHOW,false);
//--- 初始化随机数生成器
   MathSrand(GetTickCount());
//--- 生成一个随机变量样本
   long chart=0;
   string name="GraphicNormal";
   int n=100000;        // 样本值的数量
   int ncells=13;       // 直方图区间数
   double x[];          // 直方图的区间中心
   double y[];          // 区间内样本数量值下降
   double data[];       // 随机值样本
   double max,min;      // 样本的最大最小值
//--- 获得泊松分布的样本
   MathRandomPoisson(lambda_par,n,data);
//--- 计算数据来绘制直方图
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- 获得序列边界和绘制理论曲线的步骤
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- 获得[min,max]区间的理论计算数据
   double x2[];
   double y2[];
   MathSequence(0,int(MathCeil(max)),1,x2);
   MathProbabilityDensityPoisson(x2,lambda_par,false,y2);
//--- 设定比例
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- 输出图表
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Poisson distribution lambda=%G",lambda_par));
   graphic.BackgroundMainSize(16);
//--- 禁止自动缩放Y轴
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(NormalizeDouble(theor_max,2));
   graphic.YAxis().Min(0);
//--- 绘制全部曲线
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- 现在绘制分布密度的理论曲线
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- 绘制全部曲线
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  计算数据集的频率                                                |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- 定义区间中心
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- 填充区间内下降的频率
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  计算序列生成值                                                  |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- 计算序列的绝对范围获得标准化精度
   double range=MathAbs(maxv-minv);
```

© 2000-2025, MetaQuotes Ltd.

```
   int degree=(int)MathRound(MathLog10(range));
//--- 标准化指定精度的最大和最小值
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- 也基于指定精度设置序列生成步骤
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityPoisson

通过lambda参数，计算随机变量x的泊松分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityPoisson(
   const double  x,              // 随机变量值 整数）
   const double  lambda,         // 分布参数 （平均数）
   const bool    log_mode,       // 计算数值对数，如果log_mode=true，则计算概率密度的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过lambda参数，计算随机变量x的泊松分布下概率质量函数值。出错情况下它返回NaN。

```
double  MathProbabilityDensityPoisson(
   const double  x,              // 随机变量值 整数）
   const double  lambda,         // 分布参数 （平均数）
   int&          error_code      // 存储错误代码的变量
   );
```

通过lambda参数，计算随机变量数组x[]的泊松分布下概率质量函数值。出错情况下它返回false。R语言的dhyper()模拟。

```
bool  MathProbabilityDensityPoisson(
   const double& x[],            // 随机变量值数组
   const double  lambda,         // 分布参数 （平均数）
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率密度的自然
   double&       result[]        //概率密度函数值数组
   );
```

通过lambda参数，计算随机变量数组x[]的泊松分布下概率质量函数值。出错情况下它返回false。

```
bool  MathProbabilityDensityPoisson(
   const double& x[],            // 随机变量值数组
   const double  lambda,         // 分布参数 （平均数）
   double&       result[]        //概率密度函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*lambda*

[in] 分布参数 （平均数）。

*log_mode*

[in] 计算数值对数的标识。如果log_mode=true，则返回概率密度的自然对数。

*error_code*

© 2000-2025, MetaQuotes Ltd.

[out] 存储错误代码的变量。

*result[]*

[out] 概率密度函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionPoisson

通过lambda参数，计算随机变量x的泊松分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionPoisson(
   const double  x,              // 随机变量值 整数）
   const double  lambda,         // 分布参数 平均数）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       // 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   int&          error_code      // 存储错误代码的变量
   );
```

通过lambda参数，计算随机变量x的泊松分布函数值。出错情况下它返回NaN。

```
double  MathCumulativeDistributionPoisson(
   const double  x,              // 随机变量值 整数）
   const double  lambda,         // 分布参数 平均数）
   int&          error_code      // 存储错误代码的变量
   );
```

通过lambda参数，计算随机变量数组x[]的泊松分布函数值。出错情况下它返回false。R语言的dhyper()模拟。

```
bool  MathCumulativeDistributionPoisson(
   const double& x[],            // 随机变量值数组
   const double  lambda,         // 分布参数 平均数）
   const bool    tail,           // 计算标识，如果true，则计算不超过随机变量x的概率
   const bool    log_mode,       //计算数值对数的标识，如果log_mode=true，则计算概率的自然对数
   double&       result[]        // 分布函数值数组
   );
```

通过lambda参数，计算随机变量数组x[]的泊松分布函数值。出错情况下它返回false。

```
bool  MathCumulativeDistributionPoisson(
   const double& x[],            // 随机变量值数组
   const double  lambda,         // 分布参数 平均数）
   double&       result[]        // 分布函数值数组
   );
```

参数

*x*

[in] 随机变量值。

*x[]*

[in] 随机变量值数组。

*lambda*

[in] 分布参数 平均数）。

*tail*

[in] 计算标识，如果true，则计算不超过随机变量x的概率。

© 2000-2025, MetaQuotes Ltd.

*log_mode*

[in] 计算数值对数的标识，如果log_mode=true，则计算概率的自然对数。

*error_code*

[out] 存储错误代码的变量。

*result[]*

[out] 分布函数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathQuantilePoisson

对于指定*probability*，函数通过lambda参数计算逆泊松分布函数值。出错情况下它返回NaN。

```
double  MathQuantilePoisson(
   const double  probability,     // 随机变量出现的概率值
   const double  lambda,          // 分布参数 (平均数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定*probability*，函数通过lambda参数计算逆泊松分布函数值。出错情况下它返回NaN。

```
double  MathQuantilePoisson(
   const double  probability,     // 随机变量出现的概率值
   const double  lambda,          // 分布参数 (平均数)
   int&          error_code       // 存储错误代码的变量
   );
```

对于指定概率值的*probability[]*数组，函数通过lambda参数计算逆泊松分布函数值。出错情况下它返回false。R语言的qhyper()模拟。

```
double  MathQuantilePoisson(
   const double& probability[],   // 随机变量概率值数组
   const double  lambda,          // 分布参数 (平均数)
   const bool    tail,            // 计算标识，如果false，则执行1.0-概率的计算
   const bool    log_mode,        //计算标识，如果log_mode=true，则执行Exp(probability)计算
   double&       result[]         // 四分位数值数组
   );
```

对于指定概率值的*probability[]*数组，函数通过lambda参数计算逆泊松分布函数值。出错情况下它返回false。

```
bool  MathQuantilePoisson(
   const double& probability[],   // 随机变量概率值数组
   const double  lambda,          // 分布参数 (平均数)
   double&       result[]         // 四分位数值数组
   );
```

参数

*概率*

[in] 随机变量值概率。

*probability[]*

[in] 随机变量概率值数组。

*lambda*

[in] 分布参数 (平均数)。

*tail*

© 2000-2025, MetaQuotes Ltd.

[in] 计算标识，如果tail=false，则计算执行1.0-概率。

*log_mode*

[in] 计算标识，如果log_mode=true，则执行Exp(probability)计算。

*error_code*

[out] 获得错误代码的变量。

*result[]*

[out] 四分位数值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRandomPoisson

通过lambda参数生成根据泊松分布定律分布的伪随机变量。出错情况下它返回NaN。

```
double  MathRandomPoisson(
   const double  lambda,          // 分布参数 (平均数)
   int&          error_code       // 存储错误代码的变量
   );
```

通过lambda参数生成根据泊松分布定律分布的伪随机变量。出错情况下它返回false。R语言的rgeom()模拟。

```
bool  MathRandomPoisson(
   const double  lambda,          // 分布参数 (平均数)
   const int     data_count,      // 所需的数据量
   double&       result[]         // 伪随机变量值数组
   );
```

参数

*lambda*

[in] 分布参数 (平均数)。

*error_code*

[out] 存储错误代码的变量。

*data_count*

[out] 所需的数据量。

*result[]*

[out] 获得伪随机变量值的数组。

© 2000-2025, MetaQuotes Ltd.

# MathMomentsPoisson

通过lambda参数计算泊松分布的前4力矩的理论数值。

```
double  MathMomentsPoisson(
   const double  lambda,          // 分布参数 (平均数)
   double&       mean,            // 平均值变量
   double&       variance,        // 方差变量
   double&       skewness,        // 偏度变量
   double&       kurtosis,        // 峰度变量
   int&          error_code       // 错误代码变量
   );
```

参数

*lambda*

[in] 分布参数 (平均数)。

*平均值*

[out] 获得平均值的变量。

*方差*

[out] 获得方差的变量。

*偏度*

[out] 获得偏度的变量。

*峰度*

[out] 获得峰度的变量。

*error_code*

[out] 获得错误代码的变量。

返回值

如果力矩成功计算返回true，否则为false。

© 2000-2025, MetaQuotes Ltd.

# 子函数

执行基础数学操作的函数组：计算不同基础的伽玛函数，beta函数，阶乘函数，指数函数，对数函数，平方根函数等等。

它们提供包括单数值（真实型和整型）和该值数组（单独数组或原始数组的输出结果）的处理能力。

| 函数 | 描述 |
| --- | --- |
| MathRandomNonZero | 返回0.0到1.0之间的浮点随机数。 |
| MathMoments | 计算数组元素的前4力矩：平均值，方差，偏度，峰度。 |
| MathPowInt | 提高一个数字到指定的整数幂。 |
| MathFactorial | 计算指定整数的阶乘。 |
| MathTrunc | 计算指定数字或数组元素的整数部分。 |
| MathRound | 四舍五入一个数字或数组到指定的小数位。 |
| MathArctan2 | 计算角度，其正切值等于[-pi, pi]范围内两个指定数字的比值。 |
| MathGamma | 计算伽玛函数值。 |
| MathGammaLog | 计算伽玛函数的对数。 |
| MathBeta | 计算beta函数值。 |
| MathBetaLog | 计算beta函数的对数。 |
| MathBetaIncomplete | 计算不完整的beta函数值。 |
| MathGammaIncomplete | 计算不完整的伽玛函数值。 |
| MathBinomialCoefficient | 计算二项式系数。 |
| MathBinomialCoefficientLog | 计算二项式系数的对数。 |
| MathHypergeometric2F2 | 计算超几何函数值。 |
| MathSequence | 生成一个基于值的序列：第一元素，最后元素，序列步骤。 |
| MathSequenceByCount | 生成一个基于值的序列：第一元素，最后元素，序列元素数。 |
| MathReplicate | 生成一个重复的序列值。 |
| MathReverse | 生成逆向元素顺序的数组值。 |
| MathIdentical | 比较两个数组值，如果全部元素匹配则返回true。 |
| MathUnique | 生成只有唯一值的数组。 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathQuickSortAscending | 按升序排列函数。 |
| MathQuickSortDescending | 按降序排列函数。 |
| MathQuickSort | 函数排序。 |
| MathOrder | 排序后，根据数组元素的顺序生成一个排列数组。 |
| MathBitwiseNot | 对数组元素计算逐位NOT运算的结果。 |
| MathBitwiseAnd | 对数组元素计算逐位AND运算的结果。 |
| MathBitwiseOr | 对数组元素计算逐位OR运算的结果。 |
| MathBitwiseXor | 对数组元素计算逐位XOR运算的结果。 |
| MathBitwiseShiftL | 对数组元素计算逐位SHL运算的结果。 |
| MathBitwiseShiftR | 对数组元素计算逐位SHR运算的结果。 |
| MathCumulativeSum | 生成累计金额的数组。 |
| MathCumulativeProduct | 生成累计产品的数组。 |
| MathCumulativeMin | 生成累计最小值的数组。 |
| MathCumulativeMax | 生成累计最大值的数组。 |
| MathSin | 计算数组元素的sin(x)函数值。 |
| MathCos | 计算数组元素的cos(x)函数值。 |
| MathTan | 计算数组元素的tan(x)函数值。 |
| MathArcsin | 计算数组元素的arcsin(x)函数值。 |
| MathArccos | 计算数组元素的arccos(x)函数值。 |
| MathArctan | 计算数组元素的arctan(x)函数值。 |
| MathSinPi | 计算数组元素的sin(pi*x)函数值。 |
| MathCosPi | 计算数组元素的cos(pi*x)函数值。 |
| MathTanPi | 计算数组元素的tan(pi*x)函数值。 |
| MathAbs | 计算数组元素的绝对值。 |
| MathCeil | 返回数组元素最近的较大整数。 |
| MathFloor | 返回数组元素最近的较小整数。 |
| MathSqrt | 计算数组元素的平方根。 |
| MathExp | 计算数组元素的exp(x)函数值。 |
| MathPow | 计算数组元素的pow(x, power)函数值。 |
| MathLog | 计算数组元素的log(x)函数值。 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathLog2 | 计算数组元素的基数2的对数。 |
| MathLog10 | 计算数组元素的基数10的对数。 |
| MathDifference | 生成y[i]=x[i+lag]-x[i]元素差异的数组。 |
| MathSample | 生成数组元素的随机样本。 |
| MathTukeySummary | 计算数组元素的Tukey's five-number概要。 |
| MathRange | 计算数组元素的最大值和最小值。 |
| MathMin | 返回全部数组元素的最小值。 |
| MathMax | 返回全部数组元素的最大值。 |
| MathSum | 返回数组元素的总数。 |
| MathProduct | 返回数组元素的产品。 |
| MathStandardDeviation | 返回数组元素的标准偏差。 |
| MathAverageDeviation | 计算数组元素的平均绝对偏差。 |
| MathMedian | 计算数组元素的中值。 |
| MathMean | 计算数组元素的平均值。 |
| MathVariance | 计算数组元素的方差。 |
| MathSkewness | 计算数组元素的偏度。 |
| MathKurtosis | 计算数组元素的峰度。 |
| MathLog1p | 计算数组元素的log(1+x)函数值。 |
| MathExpm1 | 计算数组元素的exp(x)-1函数值。 |
| MathSinh | 计算数组元素的sinh(x) 函数值。 |
| MathCosh | 计算数组元素的cosh(x) 函数值。 |
| MathTanh | 计算数组元素的tanh(x) 函数值。 |
| MathArcsinh | 计算数组元素的arcsinh(x) 函数值。 |
| MathArccosh | 计算数组元素的arccosh(x)函数值。 |
| MathArctanh | 计算数组元素的arctanh(x)函数值。 |
| MathSignif | 四舍五入一个值到尾数指定的小数位。 |
| MathRank | 计算数组元素的级别。 |
| MathCorrelationPearson | 计算Pearson相关系数。 |
| MathCorrelationSpearman | 计算Spearman相关系数。 |
| MathCorrelationKendall | 计算Kendall相关系数。 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 描述 |
| --- | --- |
| MathQuantile | 计算符合指定概率的样本分位数。 |
| MathProbabilityDensityEmpirical | 计算随机值的经验概率密度函数。 |
| MathCumulativeDistributionEmpirical | 计算随机值的经验累积分布函数。 |

© 2000-2025, MetaQuotes Ltd.

# MathRandomNonZero

返回0.0到1.0之间的浮点随机数。

```
double  MathRandomNonZero()
```

返回值

0.0到1.0之间的浮点随机数。

© 2000-2025, MetaQuotes Ltd.

# MathMoments

计算数组元素的前4力矩：平均值，方差，偏度，峰度。

```
bool  MathMoments(
   const double&  array[],            // 数值数组
   double&        mean,               // 平均值变量
   double&        variance,           // 方差变量
   double&        skewness,           // 偏度变量
   double&        kurtosis,           // 峰度变量
   const int      start=0,            // 初始计算索引
   const int      count=WHOLE_ARRAY   // 元素数量
   )
```

参数

*array[]*

[in] 数值数组。

*平均值*

[out] 平均值变量（第1力矩）。

*方差*

[out] 方差变量（第2力矩）。

*偏度*

[out] 偏度变量（第3力矩）。

*峰度*

[out] 峰度变量（第4力矩）。

*start=0*

[in] 初始计算索引。

*count=WHOLE_ARRAY*

[in] 计算的元素数量。

返回值

如果力矩成功计算返回true，否则为False。

© 2000-2025, MetaQuotes Ltd.

# MathPowInt

提高一个数字到指定的整数幂。

```
double  MathPowInt(
   const double  x,       // 数值
   const int     power    // 提高能力
   )
```

参数

*x*

[in] 将要提高能力的双精度浮点数。

*power*

[in] 指定能力的整型。

返回值

数字**x**，提高到指定能力。

© 2000-2025, MetaQuotes Ltd.

# MathFactorial

计算指定整数的阶乘。

```
double  MathFactorial(
   const int  n   // 数值
   )
```

参数

*n*

[in] 将被计算阶乘的整数。

返回值

数值阶乘。

© 2000-2025, MetaQuotes Ltd.

# MathTrunc

计算指定数字或数组元素的整数部分。

处理双精度浮点数的版本：

```
double  MathTrunc(
   const double   x          // 数值
   )
```

返回值

指定数字的整数部分。

处理双精度浮点数组的版本。结果输出到一个新数组：

```
bool  MathTrunc(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

返回值

如果成功返回true，否则返回false。

处理双精度浮点数组的版本。结果输出到一个原数组：

```
bool  MathTrunc(
   double&        array[]    // 数值数组
   )
```

返回值

如果成功返回true，否则返回false。

参数

*x*

[in] 双精度浮点数，将会获得其整数部分。

*array[]*

[in] 双精度浮点数组，将会获得其整数部分。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

© 2000-2025, MetaQuotes Ltd.

# MathRound

四舍五入一个双精度浮点数或这类数组到指定的小数位。

四舍五入一个双精度浮点数到指定小数位的版本：

```
double  MathRound(
   const double   x,          // 数值
   const int      digits      // 小数位数
   )
```

返回值

最接近x参数的数字，其小数位数等于数位。

四舍五入一个双精度浮点数组到指定小数位的版本。结果输出到一个新数组。

```
bool  MathRound(
   const double&  array[],    // 数值数组
   int            digits,     // 小数位数
   double&        result[]    // 结果数组
   )
```

返回值

如果成功返回true，否则返回false。

四舍五入一个双精度浮点数组到指定小数位的版本。结果输出到原数组。

```
bool  MathRound(
   double&        array[],    // 数值数组
   int            digits      // 小数位数
   )
```

返回值

如果成功返回true，否则返回false。

参数

*x*

[in] 四舍五入的双精度浮点数。

*digits*

[in] 返回值的小数位数。

*array[]*

[in] 四舍五入的双精度浮点数组。

*array[]*

[out] 输出值数组。

*result[]*

© 2000-2025, MetaQuotes Ltd.

[out] 输出值数组。

© 2000-2025, MetaQuotes Ltd.

# MathArctan2

返回两个参数(x, y)的商数的反正切。

处理两个指定数(x, y)比值的版本：

```
double  MathArctan2(
   const double       y,          // Y 坐标
   const double       x           // X 坐标
   )
```

返回值

角 θ，弧度测量，-π≤θ≤π 及 tan (θ) = y 或 x，这里(x, y)是笛卡尔坐标系统的一个点。

处理x和y数组的元素组比值的版本：

```
bool  MathArctan2(
   const double&      x[],        // 数值数组
   const double&      y[],        // 数值数组
   double&            result[]    // 结果数组
   )
```

返回值

如果成功返回true，否则返回false。

参数

*y*

[in] 点的Y坐标。

*x*

[in] 点的X坐标。

*x[]*

[in] 点的X坐标数组。

*y[]*

[in] 点的Y坐标数组。

*result[]*

[out] 输出结果的数组

注释

请备注以下内容。

- 象限1的(x, y)，返回值将为：0 < θ< π/2。
- 象限2的(x, y)，返回值将为： π/2 < θ≤π。
- 象限3的(x, y)，返回值将为： -π< θ<-π/2。
- 象限4的(x, y)，返回值将为： -π/2 < θ< 0。

© 2000-2025, MetaQuotes Ltd.

这些象限外的点的返回值表示如下。

- y 等于 0 而 x 不是负数，那么 θ= 0。
- 如果 y 等于 0 而x 为负数，那么 θ= π。
- 如果 y 等于正数，而x 等于 0，那么 θ= π/2。
- 如果 y 等于负数而x 等于 0，那么 θ= -π/2。
- 如果y 等于 0 而x 也等于 0，那么 θ= -π/2。

如果 x 或 y 的参数值等于NaN，或者 x 和 y 的参数值等于正值或负值，该方法返回NaN值。

© 2000-2025, MetaQuotes Ltd.

# MathGamma

计算真实参数x的伽玛函数值。

```
double  MathGamma(
   const double  x       // 函数的参数
   )
```

参数

x

[in] 函数的真实参数。

返回值

伽玛函数值。

© 2000-2025, MetaQuotes Ltd.

# MathGammaLog

计算真实参数 x 的伽玛函数对数。

```
double  MathGammaLog(
   const double  x      // 函数的参数
   )
```

参数

*x*

[in] 函数的真实参数。

返回值

函数对数。

© 2000-2025, MetaQuotes Ltd.

# MathBeta

计算真实参数a和b的beta函数值。

```
double  MathBeta(
   const double  a,       // 函数的第一参数
   const double  b        // 函数的第二参数
   )
```

参数

*a*

[in] 函数的a参数。

*b*

[in] 函数的b参数。

返回值

函数值。

© 2000-2025, MetaQuotes Ltd.

# MathBetaLog

计算真实参数a和b的beta函数对数。

```
double  MathBetaLog(
   const double  a,       // 函数的第一参数
   const double  b        // 函数的第二参数
   )
```

参数

*a*

[in] 函数的a参数。

*b*

[in] 函数的b参数。

返回值

函数对数。

© 2000-2025, MetaQuotes Ltd.

# MathBetaIncomplete

计算不完整的beta函数值。

```
double  MathBetaIncomplete(
   const double  x,       // 函数的参数
   const double  p,       // 函数的第一参数
   const double  q        // 函数的第二参数
   )
```

参数

*x*

[in] 函数的参数。

*p*

[in] beta函数的第一参数，必须 >0.0。

*q*

[in] beta函数的第二参数，必须 >0.0。

返回值

函数值。

© 2000-2025, MetaQuotes Ltd.

# MathGammaIncomplete

计算不完整的伽玛函数值。

```
double  MathGammaIncomplete(
   double  x,           // 函数的参数
   double  alpha        // 函数的参数
   )
```

参数

*x*

[in] 函数的参数。

*alpha*

[in] 不完整的伽玛函数参数。

返回值

函数值。

© 2000-2025, MetaQuotes Ltd.

# MathBinomialCoefficient

计算二项式系数：C(n,k)=n!/(k!*(n-k)!)。

```
long  MathBinomialCoefficient(
   const int  n,       // 元素总数
   const int  k        // 组合的元素数
   )
```

参数

*n*

[in] 元素数量。

*k*

[in] 每个组合的元素数量。

返回值

N组合数选择K。

© 2000-2025, MetaQuotes Ltd.

# MathBinomialCoefficientLog

计算二项式系数的对数：Log(C(n,k))=Log(n!/(k!*(n-k)!))

整型参数版本：

```
double  MathBinomialCoefficientLog(
   const int      n,       // 元素总数
   const int      k        // 组合的元素数
   )
```

真实型参数版本：

```
double  MathBinomialCoefficientLog(
   const double   n,       // 元素的总数
   const double   k        //组合的元素数
   )
```

参数

*n*

[in] 元素数量。

*k*

[in] 每个组合的元素数量。

返回值

C(n,k)对数。

© 2000-2025, MetaQuotes Ltd.

# MathHypergeometric2F2

使用Taylor方法计算Hypergeometric_2F2 (a, b, c, d, z) 函数值。

```
double  MathHypergeometric2F2(
   const double  a,      // 函数的第一参数
   const double  b,      // 函数的第二参数
   const double  c,      // 函数的第三参数
   const double  d,      // 函数的第四参数
   const double  z       // 函数的第五参数
   )
```

参数

*a*

[in] 函数的第一参数。

*b*

[in] 函数的第二参数。

*c*

[in] 函数的第三参数。

*d*

[in] 函数的第四参数。

*z*

[in] 函数的第五参数。

返回值

函数值。

© 2000-2025, MetaQuotes Ltd.

# MathSequence

生成一个基于以下值的序列值：第一元素，最后元素，序列步骤。

处理真实值的版本：

```
bool  MathSequence(
   const double  from,        // 初始值
   const double  to,          // 最后值
   const double  step,        // 步骤
   double&       result[]     // 结果数组
   )
```

处理整型值的版本：

```
bool  MathSequence(
   const int     from,        // 初始值
   const int     to,          // 最后值
   const int     step,        // 步骤
   int&          result[]     // 结果数组
   )
```

参数

*from*

[in] 序列的第一个值

*to*

[in] 序列的最后值

*step*

[in] 序列的步骤。

*result[]*

[out] 输出序列数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSequenceByCount

生成一个基于以下值的序列值：第一元素，最后元素，序列元素数。

处理真实值的版本：

```
bool  MathSequenceByCount(
   const double  from,        // 初始值
   const double  to,          // 最后值
   const int     count,       // 记数
   double&       result[]     // 结果数组
   )
```

处理整型值的版本：

```
bool  MathSequenceByCount(
   const int     from,        // 初始值
   const int     to,          // 最后值
   const int     count,       // 记数
   int&          result[]     // 结果数组
   )
```

参数

*from*

[in] 序列的第一个值。

*to*

[in] 序列的最后值。

*count*

[in] 序列的元素数。

*result[]*

[out] 输出序列数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathReplicate

生成一个重复的序列值。

处理真实值的版本：

```
bool  MathReplicate(
   const double&  array[],    // 数值数组
   const int      count,      // 重复数
   double&        result[]    // 结果数组
   )
```

处理整型值的版本：

```
bool  MathReplicate(
   const int&     array[],    // 数值数组
   const int      count,      // 重复数
   int&           result[]    // 结果数组
   )
```

参数

*array[]*

[in] 生成序列的数组。

*count*

[in] 序列中的重复数组数量。

*result[]*

[out] 输出序列数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathReverse

生成逆向元素顺序的数组值。

处理真实值和新数组输出结果的版本：

```
bool  MathReverse(
   const double&  array[],      //  数值数组
   double&        result[]      // 结果数组
   )
```

处理整型值和新数组输出结果的版本：

```
bool  MathReverse(
   const int&     array[],      // 数值数组
   int&           result[]      // 结果数组
   )
```

处理真实值和原数组输出结果的版本。

```
bool  MathReverse(
   double&        array[]       // 数值数组
   )
```

处理整型值和原数组输出结果的版本：

```
bool  MathReverse(
   int&           array[]       // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*array[]*

[out] 输出逆向数值顺序的数组。

*result[]*

[out] 输出逆向数值顺序的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathIdentical

比较两个数组值，如果全部元素匹配则返回true。

处理真实值数组的版本：

```
bool  MathIdentical(
   const double&  array1[],      // 数值的第一数组
   const double&  array2[]       // 数值的第二数组
   )
```

处理整数值数组的版本：

```
bool  MathIdentical(
   const int&     array1[],      // 数值的第一数组
   const int&     array2[]       // 数值的第二数组
   )
```

参数

*array1[]*

[in]进行比较的第一数组。

*array2[]*

[in]进行比较的第二数组。

返回值

如果数组相等返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathUnique

生成只有唯一值的数组。

处理真实值的版本：

```
bool  MathUnique(
   const double&   array[],   // 数值数组
   double&         result[]   // 结果数组
   )
```

处理整型值的版本：

```
bool  MathUnique(
   const int&     array[],      // 数值数组
   int&           result[]      // 结果数组
   )
```

参数

*array[]*

[in] 源数组。

*result[]*

[out] 输出唯一值的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathQuickSortAscending

该函数用于通过快速排序算法同步升序排列array[]和indices[]数组。

```
void  MathQuickSortAscending(
   double&  array[],      // 数值数组
   int&     indices[],    // 索引数组
   int      first,        // 初始值
   int      last          // 最终值
   )
```

参数

*array[]*

[in][out]排序数组。

*indices[]*

[in][out]存储原数组索引的数组。

*first*

[in]开始排序的元素索引。

*last*

[in]停止排序的元素索引。

© 2000-2025, MetaQuotes Ltd.

# MathQuickSortDescending

该函数用于通过快速排序算法同步降序排列array[]和indices[]数组。

```
void  MathQuickSortDescending(
   double&  array[],      // 数值数组
   int&     indices[],    // 索引数组
   int      first,        // 初始值
   int      last          // 最终值
   )
```

参数

*array[]*

[in][out]排序数组。

*indices[]*

[in][out]存储原数组索引的数组。

*first*

[in]开始排序的元素索引。

*last*

[in]停止排序的元素索引。

© 2000-2025, MetaQuotes Ltd.

# MathQuickSort

该函数用于通过快速排序算法同步排序array[]和indices[]数组。

```
void  MathQuickSort(
   double&  array[],      // 数值数组
   int&     indices[],    // 索引数组
   int      first,        // 初始值
   int      last,         // 最终值
   int      mode          // 方向
   )
```

参数

*array[]*

[in][out]排序数组。

*indices[]*

[in][out]存储原数组索引的数组。

*first*

[in]开始排序的元素索引。

*last*

[in]停止排序的元素索引。

*mode*

[in]排序顺序（0升序，否则降序）。

© 2000-2025, MetaQuotes Ltd.

# MathOrder

排序后，根据数组元素的顺序生成一个排列整型数组。

处理真实值数组的版本：

```
bool  MathOrder(
   const double&  array[],   // 数值数组
   int&           result[]   // 结果数组
   )
```

处理整数值数组的版本：

```
bool  MathOrder(
   const int&     array[],   // 数值数组
   int&           result[]   // 结果数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出排序索引的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseNot

对数组元素计算逐位NOT运算的结果。

新数组的输出结果版本：

```
bool  MathBitwiseNot(
   const int&  array[],   // 数值数组
   int&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathBitwiseNot(
   int&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseAnd

对指定数组计算逐位AND运算的结果。

```
bool  MathBitwiseAnd(
   const int&  array1[],  // 数值的第一数组
   const int&  array2[],  // 数值的第二数组
   int&        result[]   // 结果数组
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*result[]*

[out] 输出结果的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseOr

对指定数组计算逐位OR运算的结果。

```
bool  MathBitwiseOr(
   const int&  array1[],  // 数值的第一数组
   const int&  array2[],  // 数值的第二数组
   int&        result[]   // 结果数组
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*result[]*

[out] 输出结果的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseXor

对指定数组计算逐位XOR运算的结果。

```
bool  MathBitwiseXor(
   const int&  array1[],  // 数值的第一数组
   const int&  array2[],  // 数值的第二数组
   int&        result[]   // 结果数组
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*result[]*

[out] 输出结果的数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseShiftL

对数组元素计算逐位SHL（按位向左移位）运算的结果。

新数组的输出结果版本：

```
bool  MathBitwiseShiftL(
   const int&  array[],    // 数值数组
   const int   n,          // 比移值
   int&        result[]    // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathBitwiseShiftL(
   int&        array[],    // 数值数组
   const int   n           // 比移值
   )
```

参数

*array[]*

[in] 数值数组。

*n*

[in] 要移动的位元数。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathBitwiseShiftR

对数组元素计算逐位SHR（按位向右移位）运算的结果。

新数组的输出结果版本：

```
bool  MathBitwiseShiftR(
   const int&  array[],    // 数值数组
   const int   n,          // 比移值
   int&        result[]    // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathBitwiseShiftR(
   int&        array[],    // 数值数组
   const int   n           // 比移值
   )
```

参数

*array[]*

[in] 数值数组。

*n*

[in] 要移动的位元数。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeSum

生成累计金额的数组。

新数组的输出结果版本：

```
bool  MathCumulativeSum(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCumulativeSum(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeProduct

生成累计产品的数组。

新数组的输出结果版本：

```
bool  MathCumulativeProduct(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCumulativeProduct(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeMin

生成累计最小值的数组。

新数组的输出结果版本：

```
bool  MathCumulativeMin(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCumulativeMin(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeMax

生成累计最大值的数组。

新数组的输出结果版本：

```
bool  MathCumulativeMax(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCumulativeMax(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSin

计算数组元素的sin(x)函数值。

新数组的输出结果版本：

```
bool  MathSin(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathSin(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCos

计算数组元素的cos(x)函数值。

新数组的输出结果版本：

```
bool  MathCos(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCos(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathTan

计算数组元素的tan(x)函数值。

新数组的输出结果版本：

```
bool  MathTan(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathTan(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArcsin

计算数组元素的arcsin(x)函数值。

新数组的输出结果版本：

```
bool  MathArcsin(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArcsin(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArccos

计算数组元素的arccos(x)函数值。

新数组的输出结果版本：

```
bool  MathArccos(
   const double&   array[],    // 数值数组
   double&         result[]    // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArccos(
   double&         array[]     // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArctan

计算数组元素的arctan(x)函数值。

新数组的输出结果版本：

```
bool  MathArctan(
   const double&   array[],   // 数值数组
   double&         result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArctan(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSinPi

计算数组元素的sin(pi*x)函数值。

新数组的输出结果版本：

```
bool  MathSinPi(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathSinPi(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCosPi

计算数组元素的cos(pi*x)函数值。

新数组的输出结果版本：

```
bool  MathCosPi(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCosPi(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathTanPi

计算数组元素的tan(pi*x)函数值。

新数组的输出结果版本：

```
bool  MathTanPi(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

新数组的输出结果版本：

```
bool  MathTanPi(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathAbs

计算数组元素的绝对值。

新数组的输出结果版本：

```
bool  MathAbs(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathAbs(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCeil

返回数组元素最近的较大整数。

新数组的输出结果版本：

```
bool  MathCeil(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCeil(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathFloor

返回数组元素最近的较小整数。

新数组的输出结果版本：

```
bool  MathFloor(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathFloor(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSqrt

计算数组元素的平方根。

新数组的输出结果版本：

```
bool  MathSqrt(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathSqrt(
   double&         array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathExp

计算数组元素的exp(x)函数值。

新数组的输出结果版本：

```
bool  MathExp(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathExp(
   double&       array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathPow

计算数组元素的pow (x, power)函数值。

新数组的输出结果版本：

```
bool  MathPow(
   const double&  array[],    // 数值数组
   const double   power,      // 能力
   double&        result[]    // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathPow(
   double&        array[],    // 数值数组
   const double   power       // 能力
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathLog

计算数组元素的log(x)函数值。

计算新数组输出结果的自然对数的版本。

```
bool  MathLog(
   const double&  array[],      // 数值数组
   double&        result[]      // 结果数组
   )
```

计算原始数组输出结果的自然对数的版本。

```
bool  MathLog(
   double&        array[]       // 数值数组
   )
```

计算新数组输出结果的指定基数对数的版本。

```
bool  MathLog(
   const double&   array[],     // 数值数组
   const double    base,        // 对数基数
   double&         result[]     // 结果数组
   )
```

计算原始数组输出结果的指定基数对数的版本。

```
bool  MathLog(
   double&         array[],     // 数值数组
   const double    base         // 对数基数
   )
```

参数

*array[]*

[in] 数值数组。

*base*

[in] 对数基数。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathLog2

计算数组元素的基数2的对数。

新数组的输出结果版本：

```
bool  MathLog2(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathLog2(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathLog10

计算数组元素的基数10的对数。

新数组的输出结果版本：

```
bool  MathLog10(
   const double&   array[],   // 数值数组
   double&         result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathLog10(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathLog1p

计算数组元素的log(1+x)函数值。

新数组的输出结果版本：

```
bool  MathLog1p(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathLog1p(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathDifference

生成y[i]=x[i+lag]-x[i]元素差异的数组。

单独生成的真实值数组的版本：

```
bool  MathDifference(
   const double  &array[],     // 数值数组
   const int     lag,          // lag
   double        &result[]     // 结果数组
   )
```

单独生成的整型值数组的版本：

```
bool  MathDifference(
   const int     &array[],     // 数值数组
   const int     lag,          // lag
   int           &result[]     // 结果数组
   )
```

迭代生成的真实值数组的版本（迭代次数在输入参数中设置）：

```
bool  MathDifference(
   const double  &array[],     // 数值数组
   const int     lag,          // lag
   const int     differences,  // 迭代次数
   double        &result[]     // 结果数组
   )
```

迭代生成的整型值数组的版本（迭代次数在输入参数中设置）：

```
bool  MathDifference(
   const int&    array[],      // 数值数组
   const int     lag,          // lag
   const int     differences,  // 迭代次数
   int&          result[]      // 结果数组
   )
```

参数

*array[]*

[in] 数值数组。

*lag*

[in] Lag参数。

*differences*

[in] 迭代次数。

*result[]*

[out] 输出结果的数组。

返回值

© 2000-2025, MetaQuotes Ltd.

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSample

生成数组元素的随机样本。

处理真实值数组的版本：

```
bool  MathSample(
   const double&  array[],          // 数值数组
   const int      count,            // 记数
   double&        result[]          // 结果数组
   )
```

处理整数值数组的版本：

```
bool  MathSample(
   const int&     array[],          // 数值数组
   const int      count,            // 记数
   int&           result[]          // 结果数组
   )
```

处理真实值数组的版本。它可以获得替换样本：

```
bool  MathSample(
   const double&  array[],          // 数值数组
   const int      count,            // 记数
   const bool     replace,          // flag
   double&        result[],         // 结果数组
   )
```

处理整数值数组的版本。它可以获得替换样本：

```
bool  MathSample(
   const int&     array[],          // 数值数组
   const int      count,            // 记数
   const bool     replace,          // 标识
   int&           result[]          // 结果数组
   )
```

处理真实值数组的版本，为其定义样本概率。

```
bool  MathSample(
   const double&  array[],          // 数值数组
   double&        probabilities[],  // 概率数组
   const int      count,            // 记数
   double&        result[]          // 结果数组
   )
```

处理整型值数组的版本，为其定义样本概率。

```
bool  MathSample(
   const int&     array[],          // 数值数组
   double&        probabilities[],  // 概率数组
```

© 2000-2025, MetaQuotes Ltd.

```
   const int      count,              // 记数
   int&           result[]            // 结果数组
   )
```

处理真实值数组的版本，为其定义样本概率。它可以获得替换样本：

```
bool  MathSample(
   const double&  array[],            // 数值数组
   double&        probabilities[],    // 概率数组
   const int      count,              // 记数
   const bool     replace,            // 标识
   double&        result[]            // 结果数组
   )
```

处理整型值数组的版本，为其定义样本概率。它可以获得替换样本：

```
bool  MathSample(
   const int&     array[],            // 数值数组
   double&        probabilities[],    // 概率数组
   const int      count,              // 记数
   const bool     replace,            // 标识
   int&           result[]            // 结果数组
   )
```

参数

*array[]*

[in] 整型值数组。

*probabilities[]*

[in] 样本元素的概率 数组。

*count*

[in] 元素的数量。

*replace*

[in] 允许替换样本的参数

*result[]*

[out] 输出结果的数组。

返回值

如果成功返回true，否则返回false。

注意

replace=true 参数允许执行随机抽样元素替换回原序列。

© 2000-2025, MetaQuotes Ltd.

# MathTukeySummary

计算数组元素的Tukey's five-number概要（最小值，下四分位数，中值，上四分位数，最大值）。

```
bool  MathTukeySummary(
   const double&  array[],        // 数值数组
   const bool     removeNAN,      // 标识
   double&        minimum,        // 最小值
   double&        lower_hinge,    // 下四分位数
   double&        median,         // 中值
   double&        upper_hinge,    // 上四分位数
   double&        maximum         // 最大值
   )
```

参数

*array[]*

[in] 真实值数组。

*removeNAN*

[in] 表示是否没有数值被删除的标识。

*minimum*

[out] 存储最小值的变量。

*lower_hinge*

[out] 存储下四分位数的变量。

*median*

[out] 存储中值的变量。

*upper_hinge*

[out] 存储上四分位数的变量。

*maximum*

[out] 存储最大值的变量。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathRange

计算数组元素的最大值和最小值。

```
bool  MathRange(
   const double&  array[],  // 数值数组
   double&        min,      // 最小值
   double&        max       // 最大值
   )
```

参数

*array[]*

[in] 数值数组。

*min*

[out] 存储最小值的变量。

*max*

[out] 存储最大值的变量。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathMin

返回全部数组元素的最小值。

```
double  MathMin(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

最小值。

© 2000-2025, MetaQuotes Ltd.

# MathMax

返回全部数组元素的最大值。

```
double  MathMax(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

最大值。

© 2000-2025, MetaQuotes Ltd.

# MathSum

返回数组元素的总数。

```
double  MathSum(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

元素的总数。

© 2000-2025, MetaQuotes Ltd.

# MathProduct

返回数组元素的产品。

```
double  MathProduct(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

元素的产品。

© 2000-2025, MetaQuotes Ltd.

# MathStandardDeviation

返回数组元素的标准偏差。

```
double  MathStandardDeviation(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

标准偏差。

© 2000-2025, MetaQuotes Ltd.

# MathAverageDeviation

函数计算数组元素的平均绝对偏差。

```
double  MathAverageDeviation(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

数组元素的平均绝对偏差。

© 2000-2025, MetaQuotes Ltd.

# MathMedian

计算数组元素的中值。

```
double  MathMedian(
   double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

中值。

© 2000-2025, MetaQuotes Ltd.

# MathMean

计算数组元素的平均值。

```
double  MathMean(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

平均值。

© 2000-2025, MetaQuotes Ltd.

# MathVariance

该函数计算数组元素的方差（第二力矩）。

```
double  MathVariance(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

方差值。

© 2000-2025, MetaQuotes Ltd.

# MathSkewness

该函数计算数组元素的偏度（第三力矩）。

```
double  MathSkewness(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

偏度。

© 2000-2025, MetaQuotes Ltd.

# MathKurtosis

该函数计算数组元素的峰度（第四力矩）。

```
double  MathKurtosis(
   const double&  array[]   // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

返回值

峰度。

© 2000-2025, MetaQuotes Ltd.

# MathExpm1

计算数组元素的exp(x)-1函数值。

新数组的输出结果版本：

```
bool  MathExpm1(
   const double&   array[],   // 数值数组
   double&         result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathExpm1(
   double&         array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSinh

计算数组元素的sinh(x)函数值。

新数组的输出结果版本：

```
bool  MathSinh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathSinh(
   double&       array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCosh

计算数组元素的cosh(x)函数值。

新数组的输出结果版本：

```
bool  MathCosh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathCosh(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathTanh

计算数组元素的tanh(x)函数值。

新数组的输出结果版本：

```
bool  MathTanh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathTanh(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArcsinh

计算数组元素的arcsinh(x)函数值。

新数组的输出结果版本：

```
bool  MathArcsinh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArcsinh(
   double&         array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArccosh

计算数组元素的arccosh(x)函数值。

新数组的输出结果版本：

```
bool  MathArccosh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArccosh(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathArctanh

计算数组元素的arctanh(x)函数值。

新数组的输出结果版本：

```
bool  MathArctanh(
   const double&  array[],   // 数值数组
   double&        result[]   // 结果数组
   )
```

原始数组的输出结果版本：

```
bool  MathArctanh(
   double&        array[]    // 数值数组
   )
```

参数

*array[]*

[in] 数值数组。

*result[]*

[out] 输出值数组。

*array[]*

[out] 输出值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathSignif

四舍五入一个值到尾数指定的小数位。

处理真实值的版本：

```
double  MathSignif(
   const double   x,          // 值
   const int      digits      // 小数位数
   )
```

返回值

四舍五入值。

处理真实值数组和单独数组输出结果的版本：

```
bool  MathSignif(
   const double&  array[],    // 数值数组
   int            digits,     // 小数位数
   double         result[]    // 结果数组
   )
```

返回值

如果成功返回true，否则返回false。

处理真实值数组和原数组输出结果的版本：

```
bool  MathSignif(
   double&        array[],    // 数值数组
   int            digits      // 小数位数
   )
```

返回值

如果成功返回true，否则返回false。

参数

*x*

[in] 四舍五入的真实值。

*digits*

[in] 小数位数。

*array[]*

[in] 真实值数组。

*array[]*

[out] 输出值数组。

*result[]*

[out] 输出值数组。

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# MathRank

计算数组元素的级别。

处理真实值数组的版本：

```
bool  MathRank(
   const double&  array[],  // 数值数组
   double&        rank[]    // 等级数组
   )
```

处理整数值数组的版本：

```
bool  MathRank(
   const int&     array[],  // 数值数组
   double&        rank[]    // 等级数组
   )
```

参数

*array[]*

[in] 数值数组。

*rank[]*

[out] 输出等级数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCorrelationPearson

计算Pearson相关系数。

处理真实值数组的版本：

```
bool  MathCorrelationPearson(
   const double&  array1[],  // 数值的第一数组
   const double&  array2[],  // 数值的第二数组
   double&        r          // 相关系数
   )
```

处理整数值数组的版本：

```
bool  MathCorrelationPearson(
   const int&     array1[],  // 数值的第一数组
   const int&     array2[],  // 数值的第二数组
   double&        r          // 相关系数
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*r*

[out] 存储相关系数的变量。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCorrelationSpearman

计算Spearman相关系数。

处理真实值数组的版本：

```
bool  MathCorrelationSpearman(
   const double&  array1[],  // 数值的第一数组
   const double&  array2[],  // 数值的第二数组
   double&        r          // 相关系数
   )
```

处理整数值数组的版本：

```
bool  MathCorrelationSpearman(
   const int&     array1[],   // 数值的第一数组
   const int&     array2[],   // 数值的第二数组
   double&        r           // 相关系数
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*r*

[out] 存储相关系数。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCorrelationKendall

计算Kendall相关系数。

处理真实值数组的版本：

```
bool  MathCorrelationKendall(
   const double&  array1[],  // 数值的第一数组
   const double&  array2[],  // 数值的第二数组
   double&        tau        // 相关系数
   )
```

处理整数值数组的版本：

```
bool  MathCorrelationKendall(
   const int&     array1[],  // 数值的第一数组
   const int&     array2[],  // 数值的第二数组
   double&        tau        // 相关系数
   )
```

参数

*array1[]*

[in] 数值的第一数组。

*array2[]*

[in] 数值的第二数组。

*tau*

[out] 存储相关系数的变量。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathQuantile

计算符合指定概率的样本分位数：Q[i](p) = (1 - gamma)*x[j] + gamma*x[j+1]

```
bool  MathQuantile(
   const double&  array[],     // 数值数组
   const double&  probs[],     // 概率数组
   double&        quantile[]   // 输出四分位数数组
   )
```

参数

*array[]*

[in] 数值数组。

*probs[]*

[in] 概率数组。

*quantile[]*

[out] 输出四分位数数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathProbabilityDensityEmpirical

该函数计算数组随机值的经验概率密度函数(pdf)。

```
bool  MathProbabilityDensityEmpirical(
   const double&  array[],   // 随机值数组
   const int      count,     // 对组数量
   double&        x[],       // x值数组
   double&        pdf[]      // pdf值的数组
   )
```

参数

*array[]*

[in] 随机值数组。

*count*

[in] (x, pdf(x)) 对数数量。

*x[]*

[out] 输出 x 值数组。

*pdf[]*

[out] 输出pdf(x)值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# MathCumulativeDistributionEmpirical

该函数计算数组随机值的经验累积分布函数(cdf)。

```
bool  MathCumulativeDistributionEmpirical(
   const double&  array[],   // 随机值数组
   const int      count,     // 对组数量
   double&        x[],       // x值数组
   double&        cdf[]      // cdf值数组
   )
```

参数

*array[]*

[in] 随机值数组。

*count*

[in] (x, cdf(x)) 对组数量。

*x[]*

[out] 输出 x 值数组。

*cdf[]*

[out] 输出cdf(x)值数组。

返回值

如果成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# 模糊是处理模糊逻辑的程序库

**模糊逻辑** 是真相标记为语言变量时的传统亚里士多德逻辑的综合体。模糊逻辑，相当于经典逻辑，在定义的模糊集上有其自己的模糊逻辑操作。模糊集和普通集有相同的操作，只是它们的计算目前更困难。我们也应该注意到模糊集组合构成一个模糊集。

设定模糊逻辑不同于经典逻辑的主要原则在于最接近真实反映和高水平主体性，这些可能导致重大的计算错误。

模糊模式（或系统）是一个基于模糊逻辑计算的数学模式。 这种模式架构适用于研究主题不太正规化以及其准确数学描述过于复杂或未知的情况。这些模式的输出值（错误模式）质量直接只取决于设定该模式的EA交易。减少错误的最佳选择就是绘制最完整最全面的模式，最后通过大型训练集的机器学习来调整。

模式架构的进程可以分为三个主要阶段：

1. 定义模式的输入和输出特性。
2. 构建知识库。
3. 选择模糊推理方法中的一种（Mamdani或Sugeno）。

第一个阶段直接影响后面的两个阶段并决定该模式的未来操作。知识库或，规则库是模糊规则类型集："if, then"，它可以定义检查对象输入和输出之间的关系。系统中的规则数量没有限制，也可以通过EA交易进行确定。模糊规则的生成格式如下：

如果规则条件，然后为规则结论。

规则条件描述对象的当前状态和规则结论 — 该条件如何影响对象。由于是通过模糊推理来决定所以不可以选择条件和结论的通览模式。

系统的每条规则都有其权值 — 这个特性定义了模式中规则的重要性。加权因子被分配到 [0, 1] 范围内的规则。可以在相关文献中找到的许多模糊模式示例中，不会指定加权数据，但并不意味着它不存在。实际上，在这种情况下，针对数据库的每条规则，权值都是固定且统一的。每个规则可以有两种词语和结论的类型：

1. 简单 — 包括一个模糊变量；
2. 复杂 — 包括多个模糊变量。

根据创建的知识库，模糊推理系统确定为模式。模糊逻辑推理就是通过使用知识库和模糊操作以模糊集的形式接收符合当前输入值的结论。模糊推理的两种主要类型是Mamdani和Sugeno。

© 2000-2025, MetaQuotes Ltd.

# 归属函数

**归属函数** 是一个允许计算普通集到模糊集随机元素的归属度的函数。因此，归属函数的论域应该处于[0, 1]范围。

在大部分情况下，归属函数是连续且单调的。

| 归属函数类 | 描述 |
| --- | --- |
| CConstantMembershipFunction | 通过坐标轴实施归属函数为平行直线的类 |
| CCompositeMembershipFunction | 实施归属函数组合的类 |
| CDifferencTwoSigmoidalMembershipFunction | 通过A1，A2，C1和C2参数以区分两个s型函数的形式实施归属函数的类 |
| CGeneralizedBellShapedMembershipFunction | 通过A，B和C参数实施广义钟型归属函数的类 |
| CNormalCombinationMembershipFunction | 通过B1，B2，Sigm a1和Sigm a2参数实施双边高斯归属函数的类 |
| CNormalMembershipFunction | 通过B和Sigm a参数实施对称高斯归属函数的类 |
| CP_ShapedMembershipFunction | 通过 A，B，C和D参数实施pi型归属函数的类 |
| CProductTwoSigmoidalMembershipFunction | 通过 A1，A2，C1和 C2参数以两个s型函数产品的形式实施归属函数的类 |
| CS_ShapedMembershipFunction | 通过A和B参数实施S-like归属函数的类 |
| CTrapezoidMembershipFunction | 通过X1，X2，X3和X4参数实施梯形归属函数的类 |
| CTriangularMembershipFunction | 通过X1，X2和X3参数实施三角形归属函数的类 |
| CSigmoidalMembershipFunction | 通过A和C参数实施s型归属函数的类 |
| CZ_ShapedMembershipFunction | 通过A和B参数实施z-like归属函数的类。 |
| IMembershipFunction | 全部归属函数类的基本类。 |

© 2000-2025, MetaQuotes Ltd.

# CConstantMembershipFunction

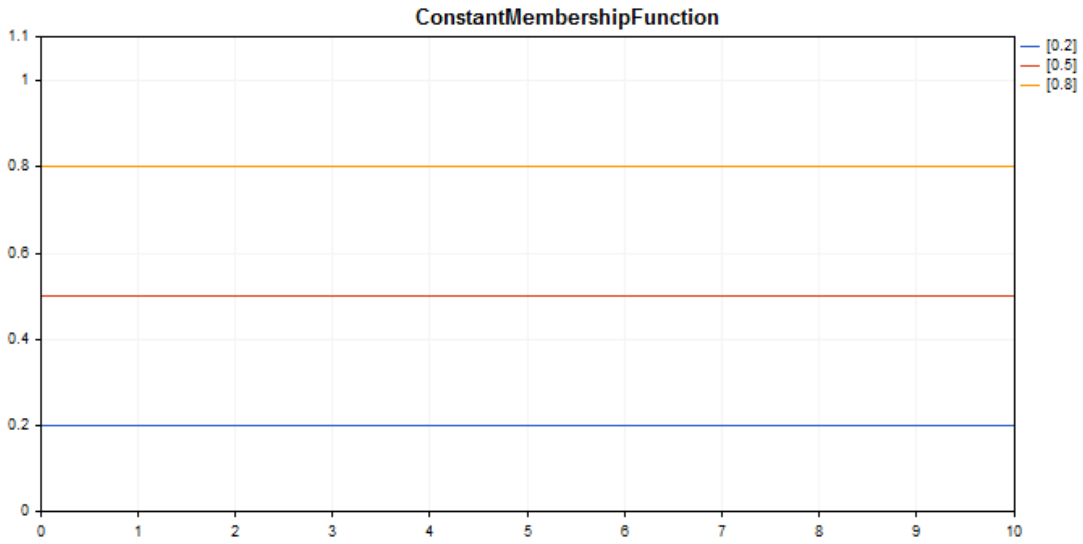
通过坐标轴实施归属函数为平行直线的类。

## 描述

该函数通过下面方程式来描述：

y(x)=c

因此，函数归属度等于整个数值轴，也等于构造函数指定的参数。




绘制图表的示例代码显示如下。

## 声明

```
class CConstantMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
IMembershipFunction
CConstantMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

© 2000-2025, MetaQuotes Ltd.

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                  ConstantMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CConstantMembershipFunction func1(0.2);
CConstantMembershipFunction func2(0.5);
CConstantMembershipFunction func3(0.8);
//--- 创建归属函数包装程序
double ConstantMembershipFunction1(double x) { return(func1.GetValue(x)); }
double ConstantMembershipFunction2(double x) { return(func2.GetValue(x)); }
double ConstantMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"ConstantMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"ConstantMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("ConstantMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(ConstantMembershipFunction1,0.0,10.0,1.0,CURVE_LINES,"[0.2]");
   graphic.CurveAdd(ConstantMembershipFunction2,0.0,10.0,1.0,CURVE_LINES,"[0.5]");
   graphic.CurveAdd(ConstantMembershipFunction3,0.0,10.0,1.0,CURVE_LINES,"[0.8]");
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.Update();
  }
```

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const double  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

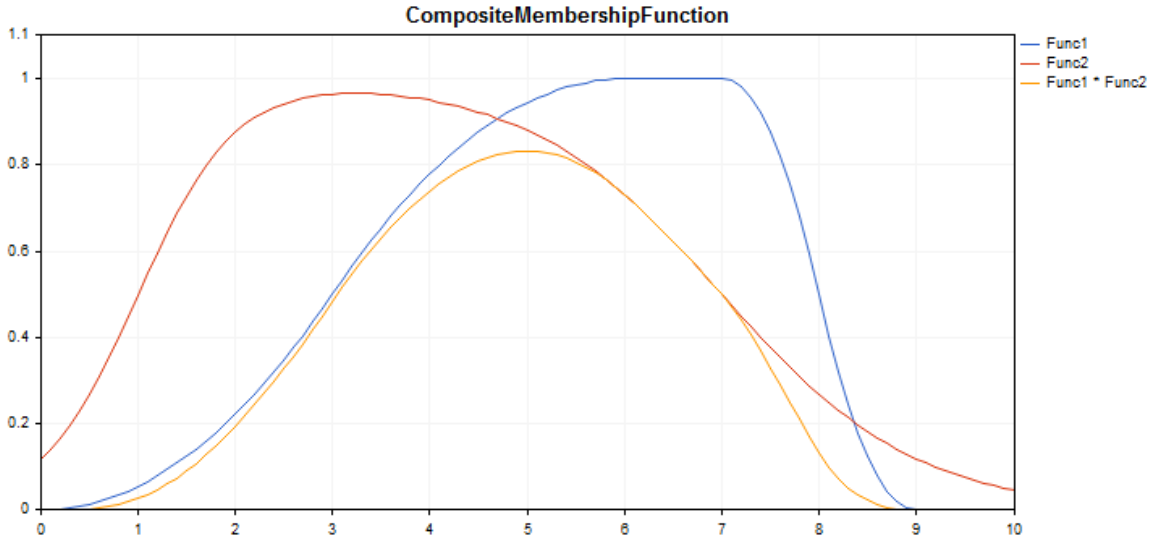
© 2000-2025, MetaQuotes Ltd.

# CCompositeMembershipFunction

实施归属函数组合的类。

## 描述

归属函数组合是使用特定操作符的两种或以上归属函数的组合。




绘制图表的示例代码显示如下。

## 声明

```
class CCompositeMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CCompositeMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| CompositionType | 设置组合操作符。 |
| MembershipFunctions | 获取归属函数列表。 |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

© 2000-2025, MetaQuotes Ltd.

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                  CompositeMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CP_ShapedMembershipFunction func2(0,6,7,9);
CCompositeMembershipFunction composite(ProdMF,GetPointer(func1),GetPointer(func2));
//--- 创建归属函数包装程序
double ProductTwoSigmoidalMembershipFunctions(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction(double x) { return(func2.GetValue(x)); }
double CompositeMembershipFunction(double x) { return(composite.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"CompositeMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"CompositeMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("CompositeMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(P_ShapedMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1");
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions,0.0,10.0,0.1,CURVE_LINES,"P
   graphic.CurveAdd(CompositeMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1 * Func
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.Update();
  }
```

## CompositionType

设置组合操作符。

```
void  CompositionType(
   MfCompositionType  value      // 操作符类型
   )
```

参数

*值*

[in] 组合操作符类型。

注意

以下操作符类型可用：

- MinMF (最小函数)
- MaxMF (最大函数)
- ProdMF (函数产品)
- SumMF (函数总数)

## MembershipFunctions

获取包含组合的归属函数列表。

```
CList*  MembershipFunctions(
   void        // 归属函数列表
   )
```

返回值

归属函数列表。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

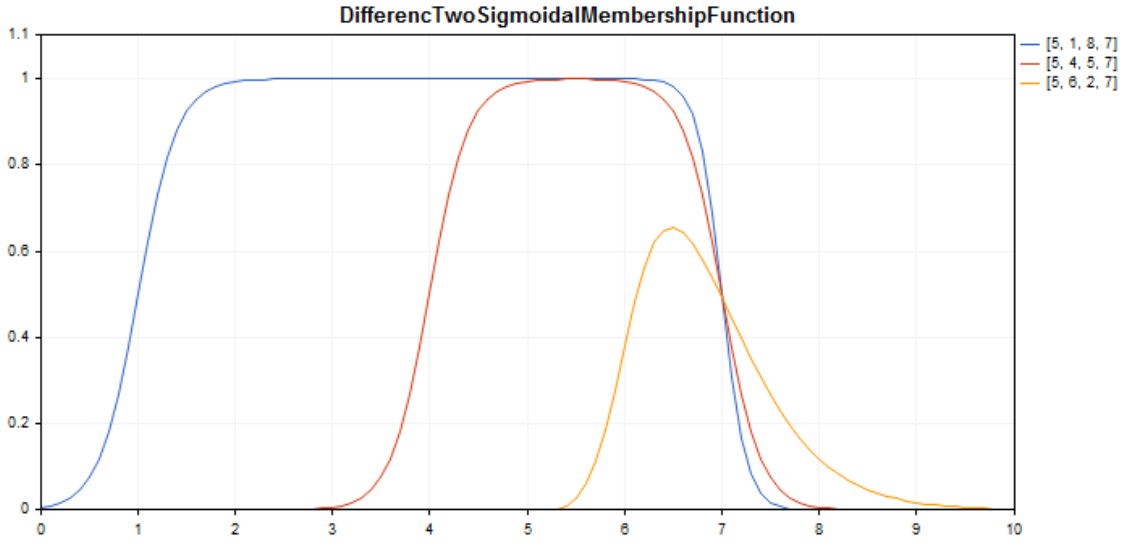
© 2000-2025, MetaQuotes Ltd.

# CDifferencTwoSigmoidalMembershipFunction

通过A1，A2，C1和C2参数以区分两个s型函数的形式实施归属函数的类。

## 描述

该函数基于s型曲线。它允许通过等于1的开始变元值创建归属函数。这类函数适用于您需要设置"short"或"long"这类语言条目的情况。




绘制图表的示例代码显示如下。

## 声明

```
class CDifferencTwoSigmoidalMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CDifferencTwoSigmoidalMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A1 | 获取和设置第一归属函数的倾斜度比率。 |
| A2 | 获取和设置第二归属函数的倾斜度比率。 |
| C1 | 获取和设置第一归属函数的变形坐标参数。 |
| C2 | 获取和设置第二归属函数的变形坐标参数。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                      DifferencTwoSigmoidalMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CDifferencTwoSigmoidalMembershipFunction func1(5,1,8,7);
CDifferencTwoSigmoidalMembershipFunction func2(5,4,5,7);
CDifferencTwoSigmoidalMembershipFunction func3(5,6,2,7);
//--- 创建归属函数包装程序
double DifferencTwoSigmoidalMembershipFunction1(double x) { return(func1.GetValue(x));
double DifferencTwoSigmoidalMembershipFunction2(double x) { return(func2.GetValue(x));
double DifferencTwoSigmoidalMembershipFunction3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"DifferencTwoSigmoidalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"DifferencTwoSigmoidalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("DifferencTwoSigmoidalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A1（获得方法）

获取第一归属函数的倾斜度比率。

```
double  A1()
```

返回值

倾斜度比率值。

## A1（设置方法）

设置第一归属函数的倾斜度比率。

```
void  A1(
   const double  a1      // 倾斜度比率值
   )
```

参数

*a1*

[in] 倾斜度比率值。

## A2（获得方法）

获取第二归属函数的倾斜度比率。

```
double  A2()
```

返回值

倾斜度比率值。

## A2（设置方法）

设置第二归属函数的倾斜度比率。

```
void  A2(
   const double  a2      // 倾斜度比率值
   )
```

参数

*a2*

[in] 倾斜度比率值。

© 2000-2025, MetaQuotes Ltd.

## C1 (获得方法)

获取第一归属函数的变形坐标参数。

```
double  C1()
```

返回值

变形坐标值。

## C1(设置方法)

设置第一归属函数的变形坐标参数。

```
void  C1(
   const double  c1      // 变形坐标值
   )
```

参数

*c1*

[in] 变形坐标值。

## C2 (获得方法)

获取第二归属函数的变形坐标参数。

```
double  C2()
```

返回值

变形坐标值。

## C2(设置方法)

设置第二归属函数的变形坐标参数。

```
void  C2(
   const double  c2      // 变形坐标值
   )
```

参数

*c2*

[in] 变形坐标值。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
```

© 2000-2025, MetaQuotes Ltd.

```
   const double  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

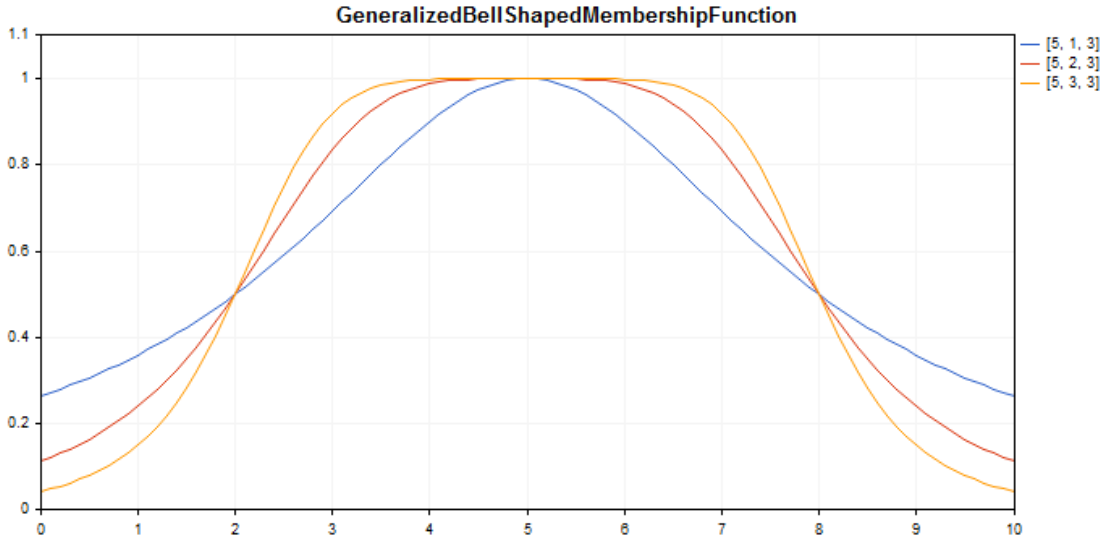
© 2000-2025, MetaQuotes Ltd.

# CGeneralizedBellShapedMembershipFunction

通过A，B和C参数实施广义钟型归属函数的类。

## 描述

广义钟型归属函数的形状类似于高斯函数。该函数是平滑型，根据整个定义区域行使非零值。




绘制图表的示例代码显示如下。

## 声明

```
class CGeneralizedBellShapedMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CGeneralizedBellShapedMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A | 获取和设置归属函数的集中度比率。 |
| B | 获取和设置归属函数的倾斜度比率。 |
| C | 获取和设置归属函数的最大坐标值。 |
| GetValue | 通过指定自变数计算归属函数值。 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                      GeneralizedBellShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CGeneralizedBellShapedMembershipFunction func1(5, 1, 3);
CGeneralizedBellShapedMembershipFunction func2(5, 2, 3);
CGeneralizedBellShapedMembershipFunction func3(5, 3, 3);
//--- 创建归属函数包装程序
double GeneralizedBellShapedMembershipFunction1(double x) { return(func1.GetValue(x));
double GeneralizedBellShapedMembershipFunction2(double x) { return(func2.GetValue(x));
double GeneralizedBellShapedMembershipFunction3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"GeneralizedBellShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"GeneralizedBellShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("GeneralizedBellShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A（获得方法）

获取归属函数的集中度比率。

```
double  A()
```

返回值

集中度比率值。

## A（设置方法）

设置归属函数的集中度比率。

```
void  A(
   const double  a      // 集中度比率值
   )
```

参数

*a*

[in] 归属函数的集中度比率。

## B（获得方法）

获取归属函数的倾斜度比率。

```
double  B()
```

返回值

倾斜度比率值。

## B（设置方法）

设置归属函数的倾斜度比率。

```
void  B(
   const double  b      // 倾斜度比率值
   )
```

参数

*b*

[in] 归属函数的倾斜度比率。

© 2000-2025, MetaQuotes Ltd.

## C（获得方法）

获取归属函数的最大坐标值。

```
double  C()
```

返回值

归属函数的最大坐标值。

## C(设置方法)

设置归属函数的最大坐标值。

```
void  C(
   const double  c      // 最大坐标值
   )
```

参数

*c*

[in] 归属函数的最大坐标值。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

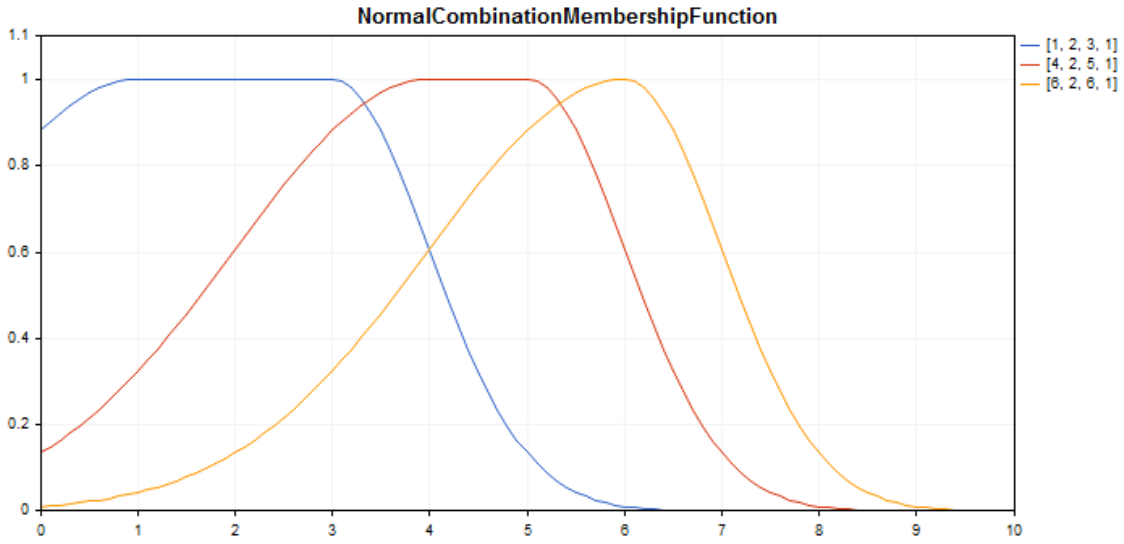
© 2000-2025, MetaQuotes Ltd.

# CNormalCombinationMembershipFunction

通过B1，B2，Sigm a1和Sigm a2参数实施双边高斯归属函数的类。

## 描述

双边高斯归属函数通过高斯分布来形成。它允许设置不对称的归属函数。该函数是平滑型，根据整个定义区域行使非零值。




绘制图表的示例代码显示如下。

## 声明

```
class CNormalCombinationMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CNormalCombinationMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| B1 | 获取和设置第一归属函数的中心值。 |
| B2 | 获取和设置第二归属函数的中心值。 |
| Sigma1 | 获取和设置归属函数曲率的第一个参数。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| Sigma2 | 获取和设置归属函数曲率的第二个参数。 |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                          NormalCombinationMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CNormalCombinationMembershipFunction func1(1,2,3,1);
CNormalCombinationMembershipFunction func2(4,2,5,1);
CNormalCombinationMembershipFunction func3(6,2,6,1);
//--- 创建归属函数包装程序
double NormalCombinationMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalCombinationMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalCombinationMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"NormalCombinationMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"NormalCombinationMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("NormalCombinationMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(NormalCombinationMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[1,
   graphic.CurveAdd(NormalCombinationMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[4,
   graphic.CurveAdd(NormalCombinationMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[6,
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## B1（获得方法）

获取第一归属函数的中心值。

```
double  B1()
```

返回值

第一归属函数的中心值。

## B1（设置方法）

设置第一归属函数的中心值。

```
void  B1(
   const double  b1      // 第一中心值
   )
```

参数

*b*

[in] 第一归属函数的中心值。

## B2（获得方法）

获取第二归属函数的中心值。

```
double  B2()
```

返回值

第二归属函数的中心值。

## B2（设置方法）

设置第二归属函数的中心值。

```
void  B2(
   const double  b2      // 第二中心值
```

© 2000-2025, MetaQuotes Ltd.

```
   )
```

参数

*b2*

[in] 第二归属函数的中心值。

## Sigma1（获得方法）

获取归属函数曲率的第一个参数。

```
double  Sigma1()
```

返回值

归属函数曲率的第一个参数值。

## Sigma1（设置方法）

设置归属函数曲率的第一个参数值。

```
void  Sigma1(
   const double  sigma1      // 第一个曲率参数值
   )
```

参数

*sigma1*

[in] 归属函数曲率的第一个参数。

## Sigma2（获得方法）

获取归属函数曲率的第二个参数。

```
double  Sigma2()
```

返回值

归属函数曲率的第二个参数值。

## Sigma2（设置方法）

设置归属函数曲率的第二个参数值。

```
void  Sigma2(
   const double  sigma2      // 第二个曲率参数值
   )
```

参数

© 2000-2025, MetaQuotes Ltd.

*sigma2*

[in] 归属函数的第二个曲率参数。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

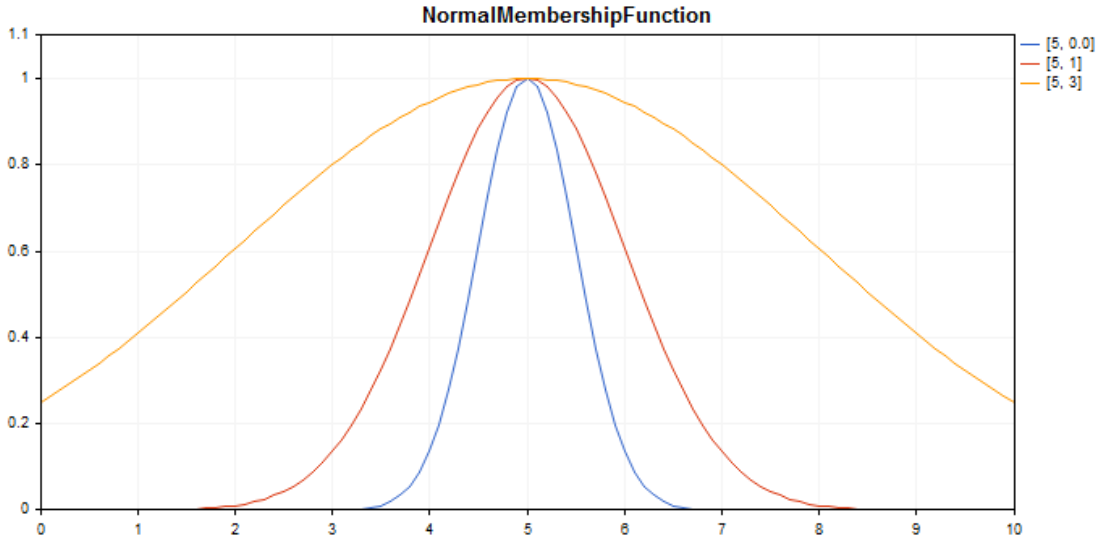
© 2000-2025, MetaQuotes Ltd.

# CNormalMembershipFunction

通过B和Sigma参数实施对称高斯归属函数的类。

## 描述

对称高斯归属函数通过高斯分布来形成。该函数是平滑型，根据整个定义区域行使非零值。




绘制图表的示例代码显示如下。

## 声明

```
class CNormalMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CNormalMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| B | 获取和设置归属函数的中心。 |
| Sigma | 获取和设置归属函数曲率的参数。 |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

© 2000-2025, MetaQuotes Ltd.

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                     NormalMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CNormalMembershipFunction func1(5,0.5);
CNormalMembershipFunction func2(5,1);
CNormalMembershipFunction func3(5,3);
//--- 创建归属函数包装程序
double NormalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"NormalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"NormalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("NormalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(NormalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[5, 0.0]");
   graphic.CurveAdd(NormalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[5, 1]");
   graphic.CurveAdd(NormalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[5, 3]");
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.Update();
  }
```

## B (获得方法)

获取归属函数的中心。

```
double  B()
```

返回值

归属函数的中心值。

## B (设置方法)

设置归属函数的中心值。

```
void  B(
   const double  b      // 函数的中心值
   )
```

参数

*b*

[in] 归属函数的中心值。

## Sigma (获得方法)

获取归属函数曲率的参数。

```
double  Sigma()
```

返回值

归属函数曲率的参数值。

## Sigma (设置方法)

设置归属函数曲率的参数值。

```
void  Sigma(
   const double  sigma      // 曲率的参数值
   )
```

参数

*sigma*

[in] 归属函数曲率的参数。

## GetValue

通过指定自变数计算归属函数值。

© 2000-2025, MetaQuotes Ltd.

```
double  GetValue(
   const double  x      // 自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

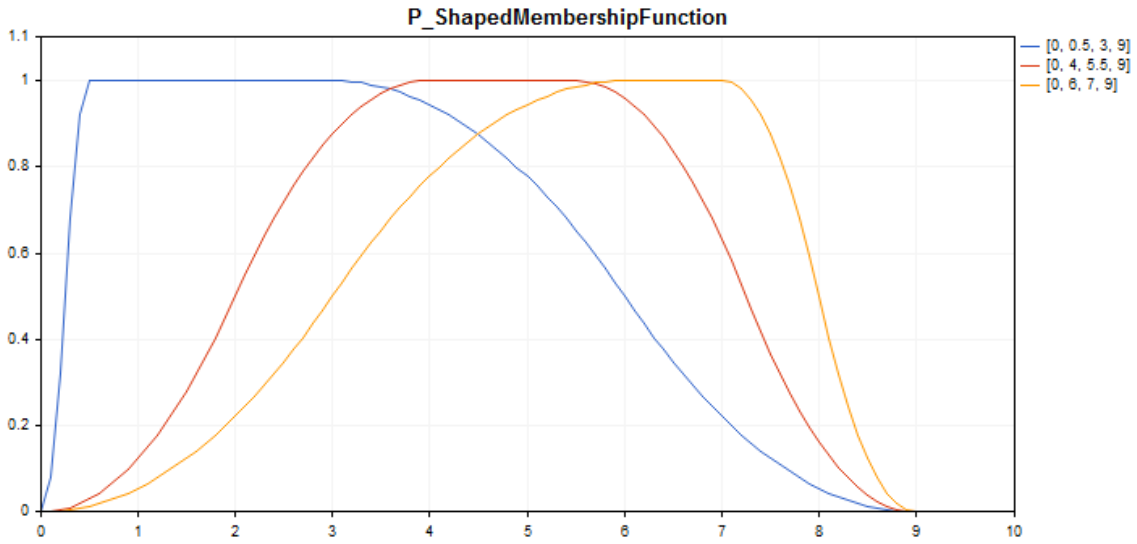
© 2000-2025, MetaQuotes Ltd.

# CP_ShapedMembershipFunction

通过A，B，C和D参数实施p型归属函数的类。

## 描述

p型归属函数具有曲线梯形的形式。该函数用于通过从悲观到乐观模糊数评估的顺利过渡设置不对称归属函数。




绘制图表的示例代码显示如下。

## 声明

```
class CP_ShapedMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CP_ShapedMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A | 获取和设置模糊集的起始参数。 |
| B | 获取和设置模糊集核心的第一参数。 |
| C | 获取和设置模糊集核心的第二参数。 |
| D | 获取和设置模糊集的结束参数。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                    P_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CP_ShapedMembershipFunction func1(0,0.5,3,9);
CP_ShapedMembershipFunction func2(0,4,5.5,9);
CP_ShapedMembershipFunction func3(0,6,7,9);
//--- 创建归属函数包装程序
double P_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double P_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"P_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"P_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("P_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(P_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 0.5, 3,
   graphic.CurveAdd(P_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 4, 5.5,
   graphic.CurveAdd(P_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[0, 6, 7, 9
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A（获得方法）

获取模糊集的起始参数。

```
double  A()
```

返回值

模糊集的起始参数。

## A（设置方法）

设置模糊集的起始参数。

```
void  A(
   const double  a      // 模糊集的起始参数
   )
```

参数

*a*

[in] 模糊集的起始参数。

## B(获得方法)

获取模糊集核心的第一参数。

```
double  B()
```

返回值

模糊集核心的第一参数。

## B(设置方法)

设置模糊集核心的第一参数。

```
void  B(
   const double  b      // 模糊集核心的第一参数值
   )
```

参数

*b*

[in] 模糊集核心的第一参数。

© 2000-2025, MetaQuotes Ltd.

## C (获得方法)

获取模糊集核心的第二参数。

```
double  C()
```

返回值

模糊集核心的第二参数。

## C (设置方法)

设置模糊集核心的第二参数。

```
void  C(
   const double  c      // 模糊集核心的第二参数值
   )
```

参数

*c*

[in] 模糊集核心的第二参数。

## D (获得方法)

获取模糊集的结束参数。

```
double  D()
```

返回值

模糊集的结束参数值。

## D (设置方法)

设置模糊集的结束参数。

```
void  D(
   const double  d      // 模糊集的结束参数值
   )
```

参数

*d*

[in] 模糊集的结束参数值。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const double  x
   )
```

© 2000-2025, MetaQuotes Ltd.

参数

*x*

[in] 归属函数自变数

返回值

归属函数值

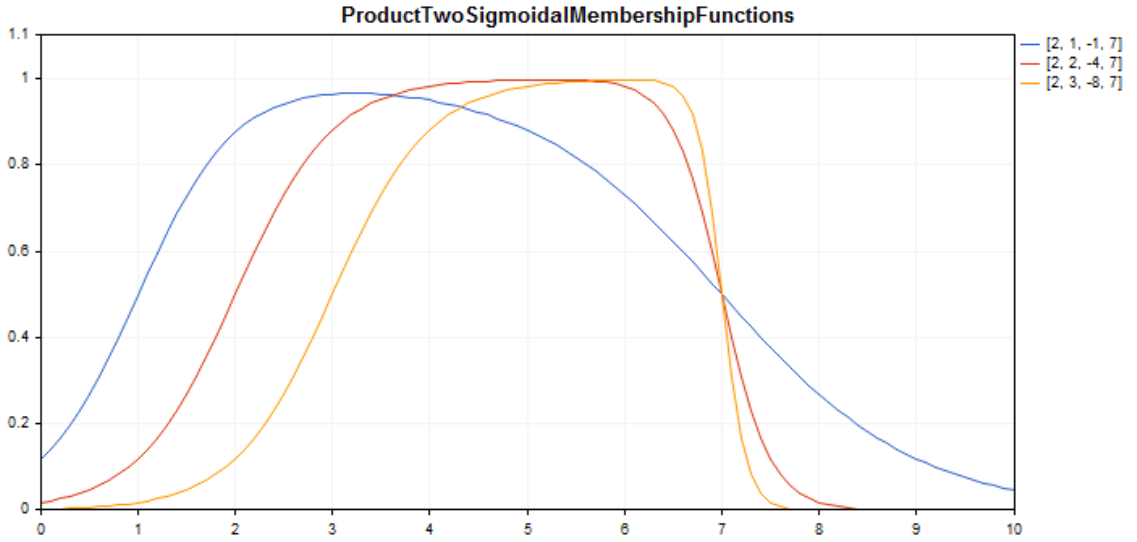
© 2000-2025, MetaQuotes Ltd.

# CProductTwoSigmoidalMembershipFunction

通过A1，A2，C1和C2参数以两个s型函数产品的形式实施归属函数的类。

## 描述

两个s型归属函数产品应用于设置平滑不对称函数。它允许通过等于1的开始变元值创建归属函数。这类函数适用于您需要设置"short"或"long"这类语言条目的情况。




绘制图表的示例代码显示如下。

## 声明

```
class CProductTwoSigmoidalMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CProductTwoSigmoidalMembershipFunctions

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A1 | 获取和设置第一归属函数的倾斜度比率。 |
| A2 | 获取和设置第二归属函数的倾斜度比率。 |
| C1 | 获取第一归属函数的变形坐标参数。 |
| C2 | 获取第二归属函数的变形坐标参数。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                      ProductTwoSigmoidalMembershipFunctions.mq5 |
//|                       Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CProductTwoSigmoidalMembershipFunctions func2(2,2,-4,7);
CProductTwoSigmoidalMembershipFunctions func3(2,3,-8,7);
//--- 创建归属函数包装程序
double ProductTwoSigmoidalMembershipFunctions1(double x) { return(func1.GetValue(x));
double ProductTwoSigmoidalMembershipFunctions2(double x) { return(func2.GetValue(x));
double ProductTwoSigmoidalMembershipFunctions3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"ProductTwoSigmoidalMembershipFunctions",0,30,30,780,380))
     {
      graphic.Attach(0,"ProductTwoSigmoidalMembershipFunctions");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("ProductTwoSigmoidalMembershipFunctions");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions1,0.0,10.0,0.1,CURVE_LINES,"
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions2,0.0,10.0,0.1,CURVE_LINES,"
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions3,0.0,10.0,0.1,CURVE_LINES,"
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A1（获得方法）

获取第一归属函数的倾斜度比率。

```
double  A1()
```

返回值

第一归属函数的倾斜度比率。

## A1（设置方法）

设置第一归属函数的倾斜度比率。

```
void  A1(
   const double  a1       // 第一归属函数的倾斜度比率
   )
```

参数

*a1*

[in] 第一归属函数的倾斜度比率。

## A2（获得方法）

获取第二归属函数的倾斜度比率。

```
double  A2()
```

返回值

第二归属函数的倾斜度比率。

## A2（设置方法）

设置第二归属函数的倾斜度比率。

```
void  A2(
   const double  a2       // 第二归属函数的倾斜度比率
   )
```

参数

© 2000-2025, MetaQuotes Ltd.

*a2*

[in] 第二归属函数的倾斜度比率。

## C1(获得方法)

获取第一归属函数的变形坐标参数。

```
double  C1()
```

返回值

第一归属函数的变形坐标。

## C1(设置方法)

设置第一归属函数的变形坐标。

```
void  C1(
   const double  c1       //第一归属函数的变形坐标
   )
```

参数

*c1*

[in] 第一归属函数的变形坐标。

## C2 (Get method)

获取第二归属函数的变形坐标参数。

```
double  C2()
```

返回值

第二归属函数的变形坐标。

## C2(设置方法)

设置第二归属函数的变形坐标。

```
void  C2(
   const double  c2       // 第二归属函数的变形坐标
   )
```

参数

*c2*

[in] 第二归属函数的变形坐标。

## GetValue

通过指定自变数计算归属函数值。

© 2000-2025, MetaQuotes Ltd.

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

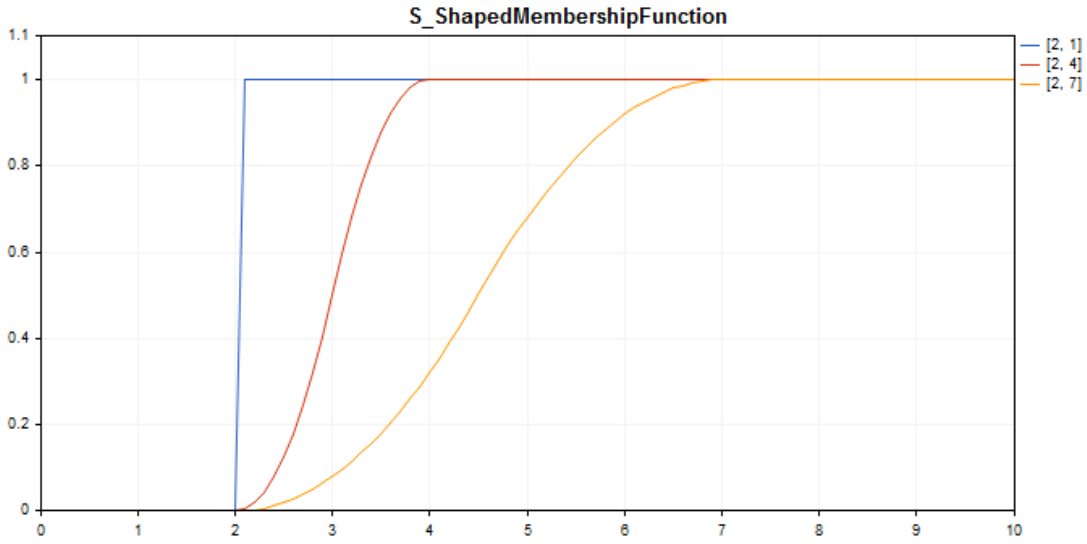
# CS_ShapedMembershipFunction

通过A和B参数实施S-like归属函数的类。

## 描述

该函数设置S-like双参数归属函数。这是一个选取0到1值的非递减函数。A和B参数定义了函数增加从0到1的非线型轨迹的间隔。

该函数表示“非常高”类型的模糊集（例如设置饱和度的非递减归属函数）。




绘制图表的示例代码显示如下。

## 声明

```
class CS_ShapedMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CS_ShapedMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A | 获取和设置增加间隔的开始参数。 |
| B | 获取和设置模糊集核心的第一参数。 |
| GetValue | 通过指定自变数计算归属函数值。 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                    S_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CS_ShapedMembershipFunction func1(2,1);
CS_ShapedMembershipFunction func2(2,4);
CS_ShapedMembershipFunction func3(2,7);
//--- 创建归属函数包装程序
double S_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double S_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double S_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"S_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"S_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("S_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(S_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
   graphic.CurveAdd(S_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
   graphic.CurveAdd(S_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 7]");
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A（获得方法）

获取增加间隔的开始参数。

```
double  A()
```

返回值

增加间隔的开始参数。

## A（设置方法）

设置增加间隔的开始参数。

```
void  A(
   const double  a      // 增加间隔的开始参数
   )
```

参数

*a*

[in] 增加间隔的开始参数。

## B（获得方法）

获取模糊集核心的第一参数。

```
double  B()
```

返回值

模糊集核心的第一参数。

## B（设置方法）

设置模糊集核心的第一参数。

```
void  B(
   const double  b      // 模糊集核心的第一参数
   )
```

参数

*b*

[in] 模糊集核心的第一参数。

## GetValue

通过指定自变数计算归属函数值。

© 2000-2025, MetaQuotes Ltd.

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

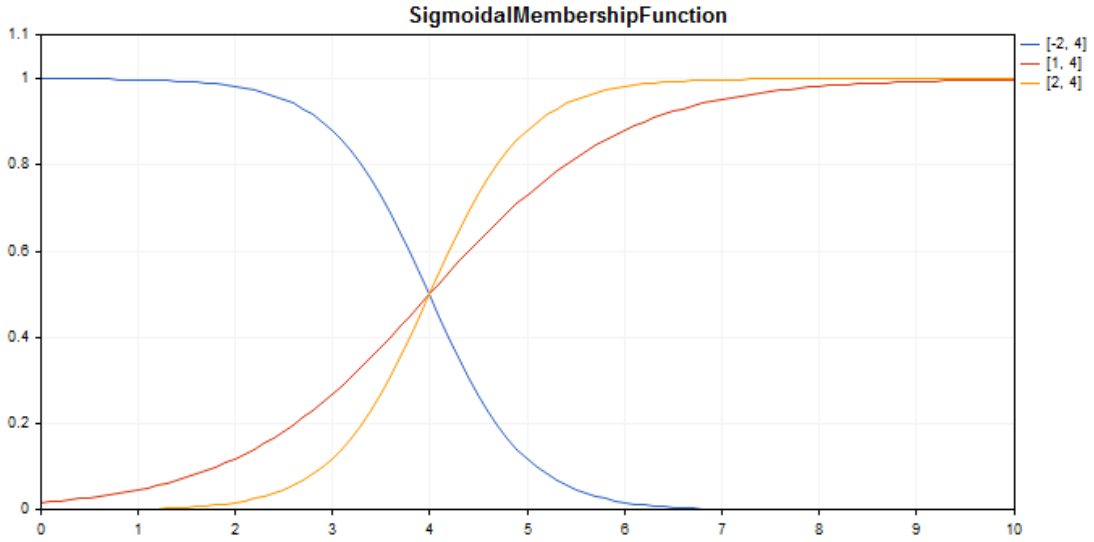
© 2000-2025, MetaQuotes Ltd.

# CSigmoidalMembershipFunction

通过A和C参数实施s型归属函数的类。

## 描述

当设置单调归属函数时使用s型函数。它允许通过等于1的开始变元值创建归属函数。这类函数适用于您需要设置"short"或"long"这类语言条目的情况。




绘制图表的示例代码显示如下。

## 声明

```
class CSigmoidalMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CSigmoidalMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A | 获取和设置归属函数的倾斜度比率。 |
| C | 获取和设置归属函数的变形坐标参数。 |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

© 2000-2025, MetaQuotes Ltd.

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                  SigmoidalMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CSigmoidalMembershipFunction func1(-2, 4);
CSigmoidalMembershipFunction func2(1, 4);
CSigmoidalMembershipFunction func3(2, 4);
//--- 创建归属函数包装程序
double SigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double SigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double SigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"SigmoidalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"SigmoidalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("SigmoidalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(SigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[-2, 4]");
   graphic.CurveAdd(SigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[1, 4]");
   graphic.CurveAdd(SigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.Update();
  }
```

## A（获得方法）

获取归属函数的倾斜度比率。

```
double  A()
```

返回值

归属函数的倾斜度比率。

## A（设置方法）

设置归属函数的倾斜度比率。

```
void  A(
   const double  a      // 第一归属函数的倾斜度比率
   )
```

参数

*a*

[in] 归属函数的倾斜度比率。

## C(获得方法)

获取归属函数的变形坐标参数。

```
double  C()
```

返回值

归属函数的变形坐标。

## C(设置方法)

设置归属函数的变形坐标。

```
void  C(
   const double  c      // 归属函数的变形坐标
   )
```

参数

*c*

[in] 归属函数的变形坐标。

## GetValue

通过指定自变数计算归属函数值。

© 2000-2025, MetaQuotes Ltd.

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

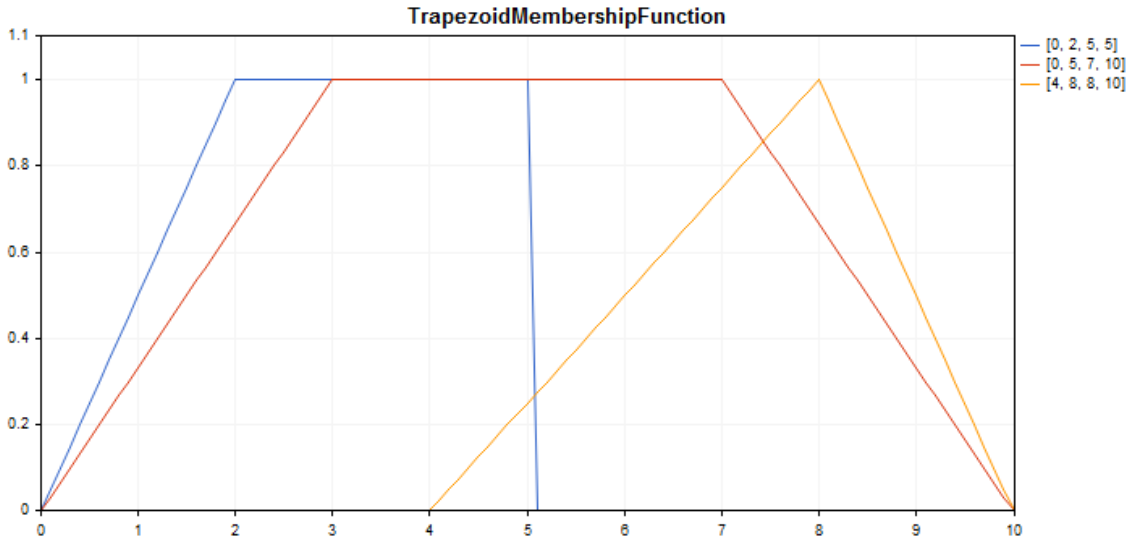
# CTrapezoidMembershipFunction

通过X1，X2，X3和X4参数实施梯形归属函数的类。

## 描述

该函数通过分段线性估算来形成。这是一个广义化三角函数，允许您指定一个模糊集核心作为一个区间。这种归属函数使得方便的解释乐观/悲观评估成为可能。

该函数用于通过一定时间间隔内定义的最关键值设置不对称的变量归属函数。




绘制图表的示例代码显示如下。

## 声明

```
class CTrapezoidMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CTrapezoidMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| X1 | 获取和设置X轴的第一个点值。 |
| X2 | 获取和设置X轴的第二个点值。 |
| X3 | 获取和设置X轴的第三个点值。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| X4 | 获取和设置X轴的第四个点值。 |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                  TrapezoidMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CTrapezoidMembershipFunction func1(0,2,5,5);
CTrapezoidMembershipFunction func2(0,3,7,10);
CTrapezoidMembershipFunction func3(4,8,8,10);
//--- 创建归属函数包装程序
double TrapezoidMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TrapezoidMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TrapezoidMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"TrapezoidMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"TrapezoidMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("TrapezoidMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(TrapezoidMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5, 5
   graphic.CurveAdd(TrapezoidMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 7, 1
   graphic.CurveAdd(TrapezoidMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[4, 8, 8, 1
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## X1（获得方法）

获取X轴的第一个点值。

```
double  X1()
```

返回值

X轴的第一个点值。

## X1（设置方法）

设置X轴的第一个点值。

```
void  X1(
   const double  x1       // X轴的第一个点值
   )
```

参数

*x1*

[in] X轴的第一个点值。

## X2（获得方法）

获取X轴的第二个点值。

```
double  X2()
```

返回值

X轴的第二个点值。

## X2（设置方法）

设置X轴的第二个点值。

```
void  X2(
   const double  x2       // X轴的第二个点值
   )
```

参数

*x2*

© 2000-2025, MetaQuotes Ltd.

[in] X轴的第二个点值。

## X3 (获得方法)

获取X轴的第三个点值。

```
double  X3()
```

返回值

X轴的第三个点值。

## X3 (设置方法)

设置X轴的第三个点值。

```
void  X3(
   const double  x3      // X轴的第三个点值
   )
```

参数

*x3*

[in] X轴的第三个点值。

## X4 (获得方法)

获取X轴的第四个点值。

```
double  X4()
```

返回值

X轴的第四个点值。

## X4 (设置方法)

设置X轴的第四个点值。

```
void  X4(
   const double  x4      // X轴的第四个点值
   )
```

参数

*x4*

[in] X轴的第四个点值。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
```

© 2000-2025, MetaQuotes Ltd.

```
   const   x       // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

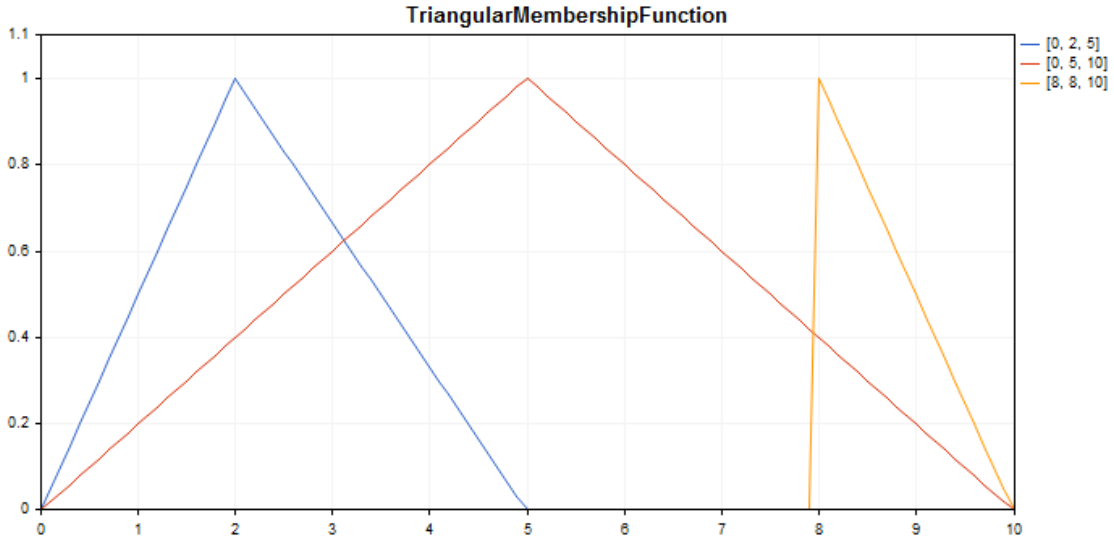
© 2000-2025, MetaQuotes Ltd.

# CTriangularMembershipFunction

通过X1，X2和X3参数实施三角形归属函数的类。

## 描述

该函数以三角形设置归属函数。这是一个简单的，使用最频繁的归属函数。




绘制图表的示例代码显示如下。

## 声明

```
class CTriangularMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CTriangularMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| X1 | 获取X轴的第一个点值。 |
| X2 | 获取X轴的第二个点值。 |
| X3 | 获取X轴的第三个点值。 |
| ToNormalMF | 将三角形归属函数转换为高斯函数。 |

© 2000-2025, MetaQuotes Ltd.

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                TriangularMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CTriangularMembershipFunction func1(0,2,5);
CTriangularMembershipFunction func2(0,5,10);
CTriangularMembershipFunction func3(8,8,10);
//--- 创建归属函数包装程序
double TriangularMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TriangularMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TriangularMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"TriangularMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"TriangularMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("TriangularMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(TriangularMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5]"
   graphic.CurveAdd(TriangularMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 10]
   graphic.CurveAdd(TriangularMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[8, 8, 10]
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## X1（获得方法）

获取X轴的第一个点值。

```
double  X1()
```

返回值

X轴的第一个点值。

## X1（设置方法）

设置X轴的第一个点值。

```
void  X1(
   const double  x1       // X轴的第一个点值
   )
```

参数

*x1*

[in] X轴的第一个点值。

## X2（获得方法）

获取X轴的第二个点值。

```
double  X2()
```

返回值

X轴的第二个点值。

## X2（设置方法）

设置X轴的第二个点值。

```
void  X2(
   const double  x2       // X轴的第二个点值
   )
```

参数

© 2000-2025, MetaQuotes Ltd.

*x2*

[in] X轴的第二个点值。

## X3 (获得方法)

获取X轴的第三个点值。

```
double  X3()
```

返回值

X轴的第三个点值。

## X3 (设置方法)

设置X轴的第三个点值。

```
void  X3(
   const double  x3      // X轴的第三个点值
   )
```

参数

*x3*

[in] X轴的第三个点值。

## ToNormalMF

将三角形归属函数转换为高斯函数。

```
CNormalMembershipFunction*  ToNormalMF()
```

返回值

高斯归属函数的指针。

## GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

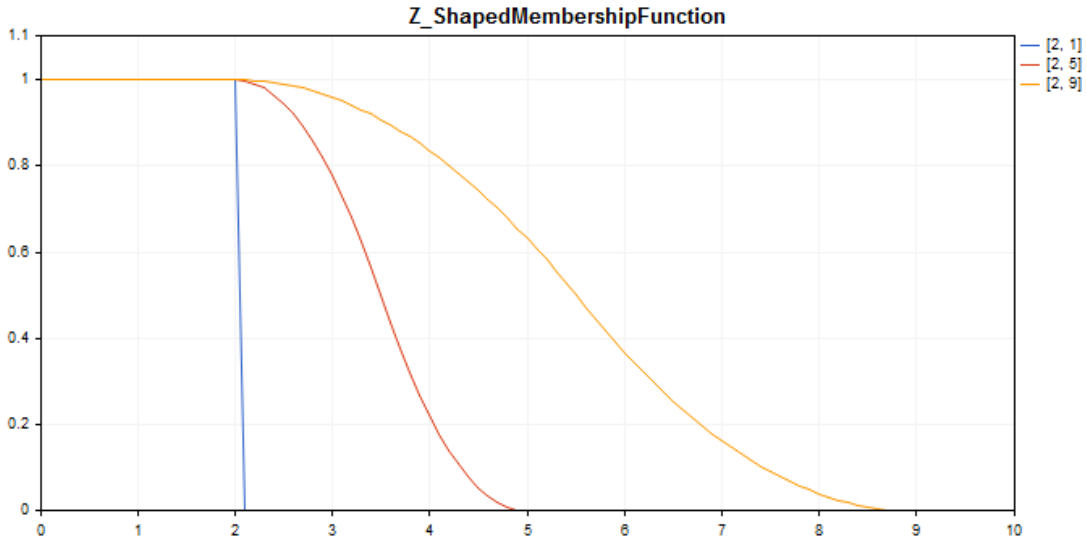
# CZ_ShapedMembershipFunction

通过A和B参数实施z-like归属函数的类。

## 描述

该函数设置z-like双参数归属函数。这是一个选取1到0值的非递增归属函数。该函数参数定义了函数递减从1到0的非线型轨迹的间隔。

该函数表示“非常低”类型的模糊集。换句话说，它通过饱和度设置了非递增归属函数。




绘制图表的示例代码显示如下。

## 声明

```
class CZ_ShapedMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject
  IMembershipFunction
    CZ_ShapedMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| A | 获取和设置减少间隔的开始参数。 |
| B | 获取和设置减少间隔的结束参数。 |
| GetValue | 通过指定自变数计算归属函数值。 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

示例

```
//+------------------------------------------------------------------+
//|                                  Z_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- 创建归属函数
CZ_ShapedMembershipFunction func1(2,1);
CZ_ShapedMembershipFunction func2(2,5);
CZ_ShapedMembershipFunction func3(2,9);
//--- 创建归属函数包装程序
double Z_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double Z_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double Z_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"Z_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"Z_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("Z_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- 创建曲线
   graphic.CurveAdd(Z_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
   graphic.CurveAdd(Z_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 5]");
   graphic.CurveAdd(Z_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 9]");
//--- 设置X轴属性
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- 设置Y轴属性
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- 绘图
```

© 2000-2025, MetaQuotes Ltd.

```
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

## A (获得方法)

获取减少间隔的开始参数。

```
double  A()
```

返回值

减少间隔的开始参数。

## A (设置方法)

设置减少间隔的开始参数。

```
void  A(
   const double  a      // 减少间隔的开始参数
   )
```

参数

*a*

[in] 减少间隔的开始参数。

## B (获得方法)

获取减少间隔的结束参数。

```
double  B()
```

返回值

减少间隔的结束参数。

## B (设置方法)

设置减少间隔的结束参数。

```
void  B(
   const double  b      // 减少间隔的结束参数
   )
```

参数

*b*

[in] 减少间隔的结束参数。

## GetValue

© 2000-2025, MetaQuotes Ltd.

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

# IMembershipFunction

全部归属函数类的基本类。

## 声明

```
class CZ_ShapedMembershipFuncion : public IMembershipFunction
```

## 主题

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

## 继承体系

CObject

IMembershipFunction

直接分支

CCompositeMembershipFunction, CConstantMembershipFunction, CDifferencTwoSigmoidalMembershipFunction, CGeneralizedBellShapedMembershipFunction, CNormalCombinationMembershipFunction, CNormalMembershipFunction, CP_ShapedMembershipFunction, CProductTwoSigmoidalMembershipFunctions, CS_ShapedMembershipFunction, CSigmoidalMembershipFunction, CTrapezoidMembershipFunction, CTriangularMembershipFunction, CZ_ShapedMembershipFunction

## 类方法

| 类方法 | 描述 |
| --- | --- |
| GetValue | 通过指定自变数计算归属函数值。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

# GetValue

通过指定自变数计算归属函数值。

```
double  GetValue(
   const  x      // 归属函数自变数
   )
```

参数

*x*

[in] 归属函数自变数。

返回值

归属函数值。

© 2000-2025, MetaQuotes Ltd.

# 模糊系统规则

**模糊系统**（模糊逻辑推理系统）就是通过使用**模糊规则**集和模糊操作集以模糊集的形式接收符合当前输入值的结论。

**模糊规则**确定检查对象输入和输出之间的关系。系统中的规则数量没有限制。模糊规则的生成格式如下：

如果 *规则条件*，然后 *规则结论*。

*规则条件*描述对象的当前状态。*规则条件*描述条件如何影响对象。

| 模糊系统的规则类 | 描述 |
| --- | --- |
| CMamdaniFuzzyRule | 通过Mamdani算法实施模糊逻辑规则的类 |
| CSugenoFuzzyRule | 通过Sugeno算法实施模糊逻辑规则的类 |
| CSingleCondition | 该类设置通过“模糊变量—模糊词语”组合表达模糊条件。 |
| CConditions | 该类通过一个操作符定义彼此相关的一组模糊条件。 |
| CGenericFuzzyRule | 实施两种模糊规则类型的基类。 |

© 2000-2025, MetaQuotes Ltd.

# CMamdaniFuzzyRule

Mamdani类型模糊推理 —模糊系统两种基本类型中的一种。输出变量值使用模糊词语设置。

## 描述

Mamdani算法的模糊逻辑规则可以描述如下：

$$if(X_1\ is\ a_1) \wedge (X_2\ is\ a_2) \wedge \ldots \wedge (X_n\ is\ a_n)\ then\ (Y\ is\ d)(W)$$

这里：

- X = (X1, X2, X3 ... Xn) —输入变量的向量；
- Y —输出变量；
- a = (a1, a2, a3 ... an) —输入变量值的向量；
- d —输出变量值；
- W —规则权值。

## 声明

```
class CMamdaniFuzzyRule : public CGenericFuzzyRule
```

## 主题

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

## 继承体系

CObject
  IParsableRule
    CGenericFuzzyRule
      CMamdaniFuzzyRule

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Conclusion | 获取和设置Mamdani模糊规则的结论 |
| Weight | 获取和设置Mamdani模糊规则的权值 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CGenericFuzzyRule

Condition, Condition, CreateCondition, CreateCondition, CreateCondition

# Conclusion (获得方法)

© 2000-2025, MetaQuotes Ltd.

获取Mamdani模糊规则的结论

```
CSingleConditon*  Conclusion()
```

返回值

Mamdani模糊规则的结论。

## Conclusion（设置方法）

设置Mamdani模糊规则的结论。

```
void  Conclusion(
   CSingleConditon*  value      // Mamdani模糊规则的结论
   )
```

参数

*值*

[in] Mamdani模糊规则的结论。

## Weight（获得方法）

获取Mamdani模糊规则的权值。

```
double  Weight()
```

返回值

Mamdani模糊规则的权值。

## Weight（设置方法）

设置Mamdani模糊规则的权值。

```
void  Weight(
   const double  value      // Mamdani模糊规则的权值
   )
```

参数

*值*

[in] Mamdani模糊规则的权值。

© 2000-2025, MetaQuotes Ltd.

# CSugenoFuzzyRule

Sugeno-类型模糊推理 — 模糊系统两种基本类型中的一种。输出变量值设置为输入变量的线性组合。

## 描述

与Mamdani规则不同，输入变量值是通过来自条目的线性函数来设置而不是通过模糊词语。Sugeno算法的模糊逻辑规则可以描述如下：

$$if(X_1\ is\ a_1)\wedge(X_2\ is\ a_2)\wedge\ldots\wedge(X_n\ is\ a_n)\ then\ (Y=b_0+b_1\cdot X_1+b_2\cdot X_2+\ldots+b_n\cdot X_n)(W)$$

这里：

- X = (X1, X2, X3 ... Xn) —输入变量的向量；
- Y —输出变量；
- a = (a1, a2, a3 ... an) —输入变量值的向量；
- b = (b1, b2, b3 ... bn) —输出值线性函数的自由词比率
- W —规则权值。

## 声明

```
class CSugenoFuzzyRule : public CGenericFuzzyRule
```

## 主题

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

## 继承体系

CObject
  IParsableRule
    CGenericFuzzyRule
      CSugenoFuzzyRule

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Conclusion | 获取和设置Sugeno模糊规则的结论 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CGenericFuzzyRule

Condition, Condition, CreateCondition, CreateCondition, CreateCondition

# Conclusion (获得方法)

© 2000-2025, MetaQuotes Ltd.

获取Sugeno模糊规则的结论。

```
CSingleConditon*  Conclusion()
```

返回值

Sugeno模糊规则的结论。

## Conclusion (设置方法)

设置Sugeno模糊规则的结论。

```
void  Conclusion(
   CSingleCondition*  value      // Sugeno模糊规则的结论
   )
```

参数

*值*

[in] Sugeno模糊规则的结论。

© 2000-2025, MetaQuotes Ltd.

# CSingleCondition

该类设置通过 "模糊变量—模糊词语"组合表达模糊条件。

## 描述

根据模糊条件，一个变量对应一个词语。模糊条件可以通过以下表达式来描述：*X等于a*，

这里：

- X是模糊变量；
- a 是模糊变量值（模糊词语）。

## 声明

```
class CSingleCondition : public ICondition
```

## 主题

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

## 继承体系

COBject

ICondition

CSingleCondition

直接分支

CFuzzyCondition

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Not | 获取和设置指明是否有必要否定该条件的标识。 |
| Term | 获取和设置该条件的模糊词语。 |
| Var | 获取和设置该条件的模糊变量。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

# Not（获得方法）

获取指明是否有必要否定该条件的标识。

```
bool Not()
```

返回值

标识值。

© 2000-2025, MetaQuotes Ltd.

## Not (设置方法)

设置指明是否有必要否定该条件的标识。

```
void  Not(
   bool  not      // 标识值
   )
```

参数

*not*

[in] 标识值。

## Term (获得方法)

获取给定条件的模糊词语。

```
INamedValue* Term()
```

返回值

给定条件的模糊词语。

## Term (设置方法)

设置给定条件的模糊词语。

```
void  Term(
   INamedValue*&  value      // 给定条件的模糊词语
   )
```

参数

*值*

[in] 给定条件的模糊词语。

## Var (获得方法)

获取给定条件的模糊变量。

```
INamedVariable* Var()
```

返回值

给定条件的模糊变量。

## Var (设置方法)

设置给定条件的模糊变量。

```
void  Var(
   INamedVariable*&  value      //给定条件的模糊变量
   )
```

© 2000-2025, MetaQuotes Ltd.

参数

*值*

[in] 模糊变量。

© 2000-2025, MetaQuotes Ltd.

# CConditions

该类通过一个操作符定义彼此相关的一组模糊条件。

## 描述

通过一个操作符彼此连接的一组模糊条件可能描述如下：

$$(X_1 \; is \; a_1) \wedge (X_2 \; is \; a_2) \wedge \ldots \wedge (X_n \; is \; a_n)$$

这里：

- X = (X1, X2, X3 ... Xn) —输入变量的向量；
- a = (a1, a2, a3 ... an) —输入变量值的向量。

在本示例中，使用*and*操作符。此外，该类中还可以使用*or*操作符。

## 声明

```
class CConditions : public ICondition
```

## 主题

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

## 继承体系

CObject
  ICondition
    CConditions

## 类方法

| 类方法 | 描述 |
| --- | --- |
| ConditionsList | 获取全部条件列表 |
| Not | 获取和设置指明是否有必要否定这些条件的标识 |
| Op | 获取和设置绑定操作符的条件类型。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

# ConditionsList

获取全部条件列表。

```
CList*  ConditionsList()
```

返回值

全部条件列表。

© 2000-2025, MetaQuotes Ltd.

## Not (获得方法)

获取指明是否有必要否定这些条件的标识。

```
bool Not()
```

返回值

标识值。

## Not (设置方法)

设置指明是否有必要否定这些条件的标识

```
void  Not(
   bool  not      // 标识值
   )
```

参数

*not*

[in] 标识值。

## Op (获得方法)

获取绑定操作符的条件类型。可以使用*and*和*or*操作符。

```
OperatorType Op()
```

返回值

绑定操作符的条件类型。

## Op (设置方法)

设置绑定操作符的条件。可以使用*and*和*or*操作符。

```
void  Op(
   OperatorType  op     // 绑定操作符的条件类型
   )
```

参数

*op*

[in] 绑定操作符的条件类型。

© 2000-2025, MetaQuotes Ltd.

# CGenericFuzzyRule

两种模糊规则类型的基类。

## 声明

```
class CGenericFuzzyRule : public IParsableRule
```

## 主题

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

## 继承体系

CObject

IParsableRule

CGenericFuzzyRule

直接分支

CMamdaniFuzzyRule, CSugenoFuzzyRule

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Conclusion | 获取和设置模糊规则的结论 |
| Condition | 获取和设置模糊规则的'if'条件（条件集） |
| CreateCondition | 通过指定参数创建模糊规则的条件 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

# Conclusion（获得方法）

获取模糊规则的结论。

```
CSingleConditon*  Conclusion()
```

返回值

模糊规则的结论。

# Conclusion（设置方法）

设置模糊规则的结论。

```
virtual void  Conclusion(
   CSingleConditon*  value      // 模糊规则的结论
   )
```

© 2000-2025, MetaQuotes Ltd.

参数

*值*

[in] 模糊规则的结论。

## Condition（获得方法）

获取模糊规则的'if'条件（条件集）。

```
CConditons*  Condition()
```

返回值

模糊条件（条件集）。

## Condition（设置方法）

设置模糊规则的'if'条件（条件集）。

```
void  Condition(
   CConditons*  value      // 模糊规则的'if'条件（条件集）
   )
```

参数

*值*

[in] 模糊条件（条件集）。

## CreateCondition

通过指定参数创建模糊规则的条件。

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable*  var,      // 模糊变量
   CFuzzyTerm*      term,     // 模糊词语
   )
```

参数

*var*

[in] 模糊变量。

*term*

[in] 模糊词语。

返回值

模糊规则状态。

## CreateCondition

通过指定参数创建模糊规则的条件。

© 2000-2025, MetaQuotes Ltd.

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable*  var,       // 模糊变量
   CFuzzyTerm*      term,      // 模糊词语
   bool             not,       // 指明是否需要否定条件的标识
   )
```

参数

*var*

[in] 模糊变量。

*term*

[in] 模糊词语。

*not*

[in] 指明是否需要否定条件的标识。

返回值

模糊规则状态。

## CreateCondition

通过指定参数创建模糊规则的条件。

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable* var,       // 模糊变量
   CFuzzyTerm*     term,      // 模糊词语
   bool            not,       // 指明是否需要否定条件的标识
   HedgeType       hedge      // 条件绑定类型
   )
```

参数

*var*

[in] 模糊变量。

*term*

[in] 模糊词语。

*not*

[in] 指明是否需要否定条件的标识。

*hedge*

[in] 条件绑定类型。

返回值

模糊规则状态。

© 2000-2025, MetaQuotes Ltd.

# 模糊系统变量

模糊（语言）变量应用于模糊系统。这些为正常语言或人工语言中词语或词语组合的变量值。语言变量组成模糊集。模糊变量的性质和数量在定义模糊集时会因每个特定任务而发生变化。

| Class | 描述 |
| --- | --- |
| CFuzzyVariable | 创建通用模糊变量的类。 |
| CSugenoVariable | 创建Sugeno型模糊变量的类。 |

© 2000-2025, MetaQuotes Ltd.

# CFuzzyVariable

创建通用模糊变量的类。

## 描述

在这里，模糊变量通过以下参数进行创建：

- 最大变量值；
- 最小变量值；
- 模糊变量名称；
- 词语集（所有可能的值，在这里可以接收语言变量）。

## 声明

```
class CFuzzyVariable : public CNamedVariableImpl
```

## 主题

```
#include <Math\Fuzzy\fuzzyvariable.mqh>
```

## 继承体系

COBject
INamedValue
INamedVariable
CNamedVariableImpl
CFuzzyVariable

## 类方法

| 类方法 | 描述 |
| --- | --- |
| AddTerm | 添加单一模糊词语到模糊变量。 |
| GetTermByName | 通过指定名称获取模糊词语。 |
| Max | 获取和设置模糊变量的最大值。 |
| Min | 获取和设置模糊变量的最小值。 |
| Terms | 获取和设置给定模糊变量的模糊词语列表。 |
| Values | 获取和设置给定模糊变量的模糊词语列表。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CNamedVariableImpl

Name, Name

© 2000-2025, MetaQuotes Ltd.

## AddTerm

添加单一模糊词语到模糊变量。

```
void  AddTerm(
   CFuzzyTerm*&  term      // 模糊词语
   )
```

参数

*term*

[in] 模糊词语。

## GetTermByName

通过指定名称获取模糊词语。

```
CFuzzyTerm*  GetTermByName(
   const string  name      // 模糊词语名称
   )
```

参数

*name*

[in] 模糊词语名称。

返回值

附有指定名称的模糊词语。

## Max（获得方法）

获取模糊变量的最大值。

```
double  Max()
```

返回值

模糊变量的最大值。

## Max（设置方法）

设置模糊变量的最大值。

```
void  Max(
   const double  max      // 模糊变量的最大值
   )
```

参数

*max*

[in] 模糊变量的最大值。

© 2000-2025, MetaQuotes Ltd.

## Min (获得方法)

获取模糊变量的最小值。

```
double  Min()
```

返回值

模糊变量的最小值。

## Max (设置方法)

设置模糊变量的最小值。

```
void  Min(
   const double  min      // 模糊变量的最小值
   )
```

参数

*min*

[in] 模糊变量的最小值。

## Terms (获得方法)

获取给定模糊变量的模糊词语列表。

```
CList*  Terms()
```

返回值

给定模糊变量的模糊词语列表。

## Terms (设置方法)

设置给定模糊变量的模糊词语列表。

```
void  Terms(
   CList*&  terms      // 给定模糊变量的模糊词语列表
   )
```

参数

*terms*

[in] 给定模糊变量的模糊词语列表。

## Values

获取给定模糊变量的模糊词语列表。

```
CList*  Values()
```

返回值

© 2000-2025, MetaQuotes Ltd.

给定变量的模糊词语列表。

© 2000-2025, MetaQuotes Ltd.

# CSugenoVariable

创建Sugeno型模糊变量的类。

## 描述

Sugeno型模糊变量不同于普通的语言变量，因为它只能通过线性函数设置而不能通过词组设置。

## 声明

```
class CSugenoVariable : public CNamedVariableImpl
```

## 主题

```
#include <Math\Fuzzy\sugenovariable.mqh>
```

## 继承体系

CObject
  INamedValue
    INamedVariable
      CNamedVariableImpl
        CSugenoVariable

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Functions | 获取Sugeno模糊变量的线性函数列表。 |
| GetFuncByName | 通过指定名称获取线性函数。 |
| Values | 获取Sugeno模糊变量的线性函数列表。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CNamedVariableImpl

Name, Name

# Functions

获取Sugeno模糊变量的线性函数列表。

```
CList*  Functions()
```

返回值

线性函数列表。

© 2000-2025, MetaQuotes Ltd.

## GetFuncByName

通过指定名称获取线性函数。

```
ISugenoFunction*  GetFuncByName(
   const string  name      // 线性函数名称
   )
```

参数

*name*

[in] 线性函数名称。

返回值

附有指定名称的线性函数。

## Values

获取Sugeno模糊变量的线性函数列表。

```
CList*  Values()
```

返回值

Sugeno模糊变量的线性函数列表。

© 2000-2025, MetaQuotes Ltd.

# CFuzzyTerm（模糊词语）

实施模糊词语的类。

## 描述

**term**是词语集的任何元素。term由两个组件定义：

- 模糊词语名称；
- 归属函数。

## 声明

```
class CFuzzyTerm : public CNamedValueImpl
```

## 主题

```
#include <Math\Fuzzy\fuzzyterm.mqh>
```

## 继承体系

CObject
  INamedValue
    CNamedValueImpl
      CFuzzyTerm

## 类方法

| 类方法 | 描述 |
| --- | --- |
| MembershipFunction | 获取模糊词语的归属函数。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CNamedValueImpl

Name, Name

© 2000-2025, MetaQuotes Ltd.

# MembershipFunction

获取模糊词语的归属函数。

```
IMembershipFunction*  MembershipFunction()
```

返回值

归属函数

© 2000-2025, MetaQuotes Ltd.

# 模糊系统

**模糊系统** (或*模糊模式*)是一个基于模糊逻辑计算的数学模式。这种模式架构适用于研究主题不太正规化以及其准确数学描述过于复杂或未知的情况。

模式架构的进程可以分为三个主要阶段：

1. 定义模式的输入和输出特性。
2. 构建知识库。
3. 选择模糊推理方法中的一种 (Mamdani和Sugeno)。

第一个阶段直接影响后面的两个阶段并决定该模式的未来操作。

**知识库**(*规则库*)就是"if, then"类型的模糊规则集，它可以定义检查对象输入和输出之间的关系。

**规则条件** 描述对象的当前状态和**规则结论** —该条件如何影响对象。

每个规则可以有两种词语和结论的类型：

1. 简单 (链接Csinglcond) —包括一个模糊变量；
2. 复杂 (链接Cconditions)—包括多个模糊变量。

系统的每条规则都有其**权值** —模式中规则的重要性。加权因子被分配到 [0, 1]范围内的规则。

根据创建的知识库，模糊推理系统确定为模式。**模糊逻辑推理** 就是通过使用知识库和模糊操作以模糊集的形式接收符合当前输入值的结论。模糊推理的两种主要类型是Mamdani和Sugeno。

© 2000-2025, MetaQuotes Ltd.

# Mamdani系统

使用模糊词语设置Mamdani系统的输出变量值。

## 描述

Mamdani算法的模糊逻辑规则可以描述如下：

$$if(X_1\ is\ a_1) \wedge (X_2\ is\ a_2) \wedge \ldots \wedge (X_n\ is\ a_n)\ then\ (Y\ is\ d)(W)$$

这里：

- X = (X1, X2, X3 ... Xn) —输入变量的向量；
- Y —输出变量；
- a = (a1, a2, a3 ... an) —输入变量值的向量；
- d —输出变量值；
- W —规则权值。

## 类方法

| 类方法 | 描述 |
|---|---|
| AggregationMethod | 设置条件聚合类型 |
| Calculate | 计算系统的模糊推理 |
| DefuzzificationMethod | 设置去模糊化方法类型 |
| EmptyRule | 基于当前系统创建一个空Mamdani模糊规则 |
| ImplicationMethod | 设置系统蕴含算子类型 |
| Output | 获取Mamdani模糊输出变量列表。 |
| OutputByName | 通过指定名称获取Mamdani模糊输出变量。 |
| ParseRule | 基于指定线条创建Mamdani模糊规则。 |
| Rules | 返回Mamdani模糊规则列表。 |

方法继承自类CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

# AggregationMethod

设置条件聚合方法类型。

```
void  AggregationMethod(
   AggregationMethod  value      // 聚合方法类型
   )
```

参数

© 2000-2025, MetaQuotes Ltd.

*值*

[in] 条件聚合方法类型。

## Calculate

计算系统的模糊推理

```
CList*  Calculate(
   CList*  inputValues      // 输入数据
   )
```

参数

*inputValues*

[in] 计算的输入数据。

返回值

计算结果。

## DefuzzificationMethod

设置去模糊化方法类型。

```
void  DefuzzificationMethod(
   DefuzzificationMethod  value      // 去模糊化方法类型。
   )
```

参数

*值*

[in] 去模糊化方法类型。

## EmptyRule

基于当前系统创建一个空Mamdani模糊规则

```
CMamdaniFuzzyRule*  EmptyRule()
```

返回值

Mamdani模糊规则。

## ImplicationMethod

设置条件蕴含算子类型。

```
void  ImplicationMethod(
   ImplicationMethod  value      // 蕴含算子类型
   )
```

参数

© 2000-2025, MetaQuotes Ltd.

*值*

[in] 条件蕴含算子类型。

## Output

获取Mamdani模糊输出变量列表。

```
CList*  Output()
```

返回值

模糊变量列表。

## OutputByName

通过指定名称获取Mamdani模糊输出变量。

```
CFuzzyVariable*  OutputByName(
   const string  name      // 模糊变量名称
   )
```

参数

*name*

[in] 模糊变量名称。

返回值

指定名称的Mamdani模糊变量。

## ParseRule

基于指定线条创建Mamdani模糊规则。

```
CMamdaniFuzzyRule*  ParseRule(
   const string  rule      // 模糊规则的字符串表示
   )
```

参数

*rule*

[in] Mamdani模糊规则的字符串表示。

返回值

Mamdani模糊规则。

## Rules

返回Mamdani模糊规则列表。

```
CList*  Rules()
```

© 2000-2025, MetaQuotes Ltd.

返回值

Mamdani模糊规则列表。

© 2000-2025, MetaQuotes Ltd.

# CSugenoFuzzyRule

Sugeno模糊逻辑系统是模糊系统两种基本类型中的一种。输出变量值设置为输入变量的线性组合。

## 描述

与Mamdani规则不同，输入变量值是通过来自条目的线性函数来设置而不是通过模糊词语。Sugeno算法的模糊逻辑规则可以描述如下：

$$if(X_1\ is\ a_1)\wedge(X_2\ is\ a_2)\wedge\ldots\wedge(X_n\ is\ a_n)\ then\ (Y=b_0+b_1\cdot X_1+b_2\cdot X_2+\ldots+b_n\cdot X_n)(W)$$

这里：

- X = (X1, X2, X3 ... Xn) —输入变量的向量；
- Y —输出变量；
- a = (a1, a2, a3 ... an) —输入变量值的向量；
- b = (b1, b2, b3 ... bn) —输出值线性函数的自由词比率
- W —规则权值。

## 类方法

| 类方法 | 描述 |
| --- | --- |
| Calculate | 计算系统的模糊推理 |
| CreateSugenoFunction | 创建系统的Sugeno线性函数。 |
| EmptyRule | 基于当前系统创建一个空Sugeno模糊规则。 |
| Output | 获取Sugeno模糊输出变量列表。 |
| OutputByName | 通过指定名称获取Sugeno模糊输出变量。 |
| ParseRule | 基于指定线条创建Sugeno模糊规则。 |
| Rules | 返回模糊规则列表。 |

方法继承自类CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

# Calculate

计算系统的模糊推理

```
CList*  Calculate(
   CList*&  inputValues      // 输入数据
   )
```

参数

*inputValues*

© 2000-2025, MetaQuotes Ltd.

[in] 计算的输入数据。

返回值

计算结果。

## CreateSugenoFunction

创建系统的Sugeno线性函数。

```
CLinearSugenoFunction*  CreateSugenoFunction(
   const string  name,          // 函数名称
   const double& coeffs[]       // 函数比率
   )
```

参数

*name*

[in] 函数名称。

*coeffs[]*

[in] 函数比率。

返回值

Sugeno线性函数。

注意

数组的比率大小可能等于输入数字或比这个数字多一。第一种情况下，Sugeno线性函数的自由词等于0，而第二种情况，它等于最后一个比率。

## CreateSugenoFunction

创建系统的Sugeno线性函数。

```
CLinearSugenoFunction*  CreateSugenoFunction(
   const string  name,          // 函数名称
   CList*&       coeffs,        // 模糊变量组列表 - 它的比率
   const double  constValue     // 函数自由词比率
   )
```

参数

*name*

[in] 函数名称。

*coeffs[]*

[in] 函数比率。

返回值

Sugeno线性函数。

© 2000-2025, MetaQuotes Ltd.

## EmptyRule

基于当前系统创建一个空Sugeno模糊规则。

```
CSugenoFuzzyRule*  EmptyRule()
```

返回值

Sugeno模糊规则。

## Output

获取Sugeno模糊输出变量列表。

```
CList*  Output()
```

返回值

模糊变量列表。

## OutputByName

通过指定名称获取Sugeno模糊输出变量。

```
CSugenoVariable*  OutputByName(
   const string  name      // 模糊变量名称
   )
```

参数

*name*

模糊变量名称。

返回值

指定名称的Sugeno模糊变量。

## ParseRule

基于指定线条创建Sugeno模糊规则。

```
CSugenoFuzzyRule*  ParseRule(
   const string  rule      // Sugeno模糊规则的字符串表示
   )
```

参数

*rule*

[in] Sugeno模糊规则的字符串表示。

返回值

Sugeno模糊规则。

© 2000-2025, MetaQuotes Ltd.

# Rules

返回模糊规则列表。

```
CList*  Rules()
```

返回值

模糊规则列表。

© 2000-2025, MetaQuotes Ltd.

# 处理OpenCL程序的类

COpenCL类是帮助处理OpenCL函数的封装器。在某些情况下，使用GPU可以大幅度提高计算速度。

对于基于浮动值和双精度值计算的类的使用示例可以在MQL5\Scripts\Examples\OpenCL\ 文件夹相应的子目录中找到。OpenCL程序的源代码位于MQL5\Scripts\Examples\OpenCL\Double\Kernels和MQL5\Scripts\Examples\OpenCL\Float\Kernels子目录中。

- MatrixMult.mq5 - 使用全局内存和本地内存的矩阵乘法示例
- BitonicSort.mq5 - GPU数组元素平行排序的示例
- FFT.mq5 - 快速傅里叶变换计算的示例
- Wavelet.mq5 - 使用Morlet小波进行的小波数据变换的示例。

建议以单独的CL文件编写OpenCL源代码，稍后将其包含进使用资源变量MQL5程序。

## 声明

```
class COpenCL
```

## 标题

```
#include <OpenCL\OpenCL.mqh>
```

## 类方法

| 名称 | 描述 |
| --- | --- |
| BufferCreate | 在指定索引创建OpenCL缓冲区 |
| BufferFree | 在指定索引删除缓冲区 |
| BufferFromArray | 在指定索引从数组值创建缓冲区 |
| BufferRead | 在指定索引读取OpenCL缓冲区到数组 |
| BufferWrite | 在指定索引编写数组值到缓冲区 |
| Execute | 在指定索引执行OpenCL内核 |
| GetContext | 返回OpenCL上下文处理 |
| GetKernel | 在指定索引返回内核对象句柄 |
| GetKernelName | 在指定索引返回内核对象名称 |
| GetProgram | 返回OpenCL程序句柄 |
| Initialize | 初始化OpenCL程序 |
| KernelCreate | 在指定索引创建OpenCL程序切入点 |
| KernelFree | 在指定索引移除OpenCL起始函数 |
| SetArgument | 在指定索引设置OpenCL函数的参数 |

© 2000-2025, MetaQuotes Ltd.

| 名称 | 描述 |
| --- | --- |
| SetArgumentBuffer | 在指定索引设置OpenCL缓冲区作为OpenCL函数的参数 |
| SetArgumentLocalMemory | 在指定索引为OpenCL函数设置本地内存的参数 |
| SetBuffersCount | 设置缓冲区数量 |
| SetKernelsCount | 设置内核对象数量 |
| Shutdown | 卸载OpenCL程序 |
| SupportDouble | 检查设备是否支持浮动点数据类型 |

© 2000-2025, MetaQuotes Ltd.

# BufferCreate

在指定索引创建OpenCL缓冲区。

```
bool  BufferCreate(
   const int   buffer_index,      // 缓冲区索引
   const uint  size_in_bytes,     // 缓冲区的字节大小
   const uint  flags              // 定义缓冲区属性的标识组合
   );
```

参数

*buffer_index*

[in] 缓冲区索引。

*size_in_bytes*

[in] 缓冲区的字节大小。

*flags*

[in] 使用标识组合设置缓冲区属性。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# BufferFree

在指定索引删除缓冲区。

```
bool  BufferFree(
   const int  buffer_index      // 缓冲区索引
   );
```

参数

*buffer_index*

[in] 缓冲区索引。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# BufferFromArray

在指定索引从数组值创建缓冲区。

```
template<typename T>
bool  BufferFromArray(
   const int   buffer_index,           // 缓冲区索引
   T           &data[],                // 数组值
   const uint  data_array_offset,      // 以字节为单位的数组值抵消
   const uint  data_array_count,       // 要编写的数组值数
   const uint  flags                   // 定义缓冲区属性的标识组合
   );
```

参数

*buffer_index*

[in] 缓冲区索引。

*&data[]*

[in] 将要写入OpenCL缓冲区的数组值。

*data_array_offset*

[in] 以字节为单位的数组值抵消，自初始值开始。

*data_array_count*

[in]将编写的值数。

*flags*

[in] 使用标识组合设置缓冲区属性。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# BufferRead

在指定索引读取OpenCL缓冲区到数组。

```
template<typename T>
bool  BufferRead(
   const int   buffer_index,          // 缓冲区索引
   T           &data[],               // 数组值
   const uint  cl_buffer_offset,      // 以字节为单位的OpenCL缓冲区抵消
   const uint  data_array_offset,     // 数组元素的变化
   const uint  data_array_count       // 将要读取的缓冲区值数
   );
```

参数

*buffer_index*

[in] 缓冲区索引。

*&data[]*

[in] 获得OpenCL缓冲区值的数组。

*cl_buffer_offset*

[in] 以字节为单位的OpenCL缓冲区抵消，并自此开始读值。

*data_array_offset*

[in] 数组第一元素索引，编写OpenCL 缓冲区值。

*data_array_count*

[in] 将读取的值数。

返回值

如果执行成功，返回true，否则 -false。

© 2000-2025, MetaQuotes Ltd.

# BufferWrite

在指定索引编写数组值到缓冲区。

```
template<typename T>
bool  BufferWrite(
   const int    buffer_index,          // 缓冲区索引
   T            &data[],               // 数组值
   const uint   cl_buffer_offset,      // 以字节为单位的OpenCL缓冲区抵消
   const uint   data_array_offset,     // 数组元素的变化
   const uint   data_array_count       // 要编写的数组值数
   );
```

参数

*buffer_index*

[in] 缓冲区索引。

*&data[]*

[in] 将要写入OpenCL缓冲区的数组值。

*cl_buffer_offset*

[in] 以字节为单位的OpenCL缓冲区抵消，并自此开始写值。

*data_array_offset*

[in] 数组第一元素索引，数组值自此开始写入OpenCL缓冲区。

*data_array_count*

[in] 将编写的值数。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Execute

在指定索引执行OpenCL程序

```
bool  Execute(
   const int   kernel_index,           // 内核索引
   const int   work_dim,               // 任务空间维度
   const uint  &work_offset[],         // 任务空间的初始抵消
   const uint  &work_size[]            // 任务总数
   );
```

执行指定索引的OpenCL内核和本地组的任务数。

```
bool  Execute(
   const int   kernel_index,           // 内核索引
   const int   work_dim,               // 任务空间维度
   const uint  &work_offset[],         // 任务空间的初始抵消
   const uint  &work_size[],           // 任务总数
   const uint  &local_work_size[]      // 本地组的任务数
   );
```

参数

*kernel_index*

[in] 内核对象索引。

*work_dim*

[in] 任务空间的维度。

*&work_offset[]*

[in][out] 任务空间的初始抵消。引用传递。

*&work_size[]*

[in][out] 任务子集的大小。引用传递。

*&local_work_size[]*

[in][out] 组中本地任务子集的大小。引用传递。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# GetContext

返回OpenCL上下文处理。

```
int  GetContext();
```

返回值

OpenCL上下文处理。

© 2000-2025, MetaQuotes Ltd.

# GetKernel

在指定索引返回内核对象句柄。

```
int  GetKernel(
   const int  kernel_index      // 内核索引
   );
```

参数

*kernel_index*

[in] 内核对象索引。

返回值

内核对象句柄。

© 2000-2025, MetaQuotes Ltd.

# GetKernelName

在指定索引返回内核对象名称。

```
string  GetKernelName(
   const int  kernel_index      // 内核索引
   );
```

参数

*kernel_index*

[in] 内核对象索引。

返回值

内核对象的名称。

© 2000-2025, MetaQuotes Ltd.

# GetProgram

返回OpenCL程序句柄。

```
int  GetProgram();
```

返回值

OpenCL程序句柄。

© 2000-2025, MetaQuotes Ltd.

# Initialize

初始化OpenCL程序。

```
bool  Initialize(
   const string  program,           // OpenCL 程序句柄
   const bool    show_log=true      // 保存日志
   );
```

参数

*程序*

[in] OpenCL程序句柄。

*show_log=true*

[in] 启用日志消息。

返回值

如果初始化成功，返回true。否则，返回false。

© 2000-2025, MetaQuotes Ltd.

# KernelCreate

在指定索引创建OpenCL程序切入点。

```
bool  KernelCreate(
   const int     kernel_index,     // 内核索引
   const string  kernel_name       // 内核名称
   );
```

参数

*kernel_index*

[in] 内核对象索引。

*kernel_name*

[in] 内核对象名称。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# KernelFree

在指定索引移除OpenCL起始函数。

```
bool  KernelFree(
   const int  kernel_index      // 内核索引
   );
```

参数

*kernel_index*

[in] 内核对象索引。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# SetArgument

在指定索引设置OpenCL函数的参数。

```
template<typename T>
bool  SetArgument(
   const int  kernel_index,      // 内核索引
   const int  arg_index,         // 函数参数索引
   T          value              // 源代码
   );
```

参数

*kernel_index*

[in] 内核对象索引。

*arg_index*

[in] 函数参数索引。

*value*

[in] 函数参数值。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# SetArgumentBuffer

在指定索引设置OpenCL缓冲区作为OpenCL函数的参数。

```
bool  SetArgumentBuffer(
   const int  kernel_index,      // 内核索引
   const int  arg_index,         // 函数参数索引
   const int  buffer_index       // 缓冲区索引
   );
```

参数

*kernel_index*

[in] 内核对象索引。

*arg_index*

[in] 函数参数索引。

*buffer_index*

[in] 缓冲区索引。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# SetArgumentLocalMemory

在指定索引为OpenCL函数设置本地内存的参数。

```
bool  SetArgumentLocalMemory(
   const int  kernel_index,          // 内核索引
   const int  arg_index,             // 函数参数索引
   const int  local_memory_size      // 本地内存大小
   );
```

参数

*kernel_index*

[in] 内核对象索引。

*arg_index*

[in] 函数参数索引。

*local_memory_size*

[in] 本地内存大小。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# SetBuffersCount

设置缓冲区数量。

```
bool  SetBuffersCount(
   const int  total_buffers      // 缓冲区数量
   );
```

参数

*total_buffers*

[in] 缓冲区总数。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# SetKernelsCount

设置内核对象数量。

```
bool  SetKernelsCount(
   const int  total_kernels      // 内核数量
   );
```

参数

*total_kernels*

[in] 内核总数。

返回值

如果执行成功，返回true，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Shutdown

卸载OpenCL程序。

```
void  Shutdown();
```

返回值

无返回值。

© 2000-2025, MetaQuotes Ltd.

# SupportDouble

检查设备是否支持浮动点数据类型。

```
bool   SupportDouble();
```

返回值

如果设备支持浮动点数据类型，则返回true。

© 2000-2025, MetaQuotes Ltd.

# 基本类CObject

类CObject是构造MQL5标准库的基准类。

## 描述

CObject类将其所有子类作为链表的一部分。同时标定一些虚函数，以便将来在子类里实现。

## 声明

```
class CObject
```

## 标称库文件

```
#include <Object.mqh>
```

## 继承体系

CObject

直接分支

CAccountInfo, CArray, CChart, CChartObject, CCurve, CDealInfo, CDictionary_Obj_Double, CDictionary_Obj_Obj, CDictionary_String_Obj, CExpertBase, CFile, CHistoryOrderInfo, CList, COrderInfo, CPositionInfo, CString, CSymbolInfo, CTerminalInfo, CTrade, CTreeNode, CWnd, ICondition, IExpression, IMembershipFunction, INamedValue, IParsableRule

## 类方法

| 属性 | |
|---|---|
| Prev | 获取之前项目的值。 |
| Prev | 设置之前项目的值。 |
| Next | 获取后续元素的值 |
| Next | 设置后续元素的值 |
| 比较方法 | |
| virtual Compare | 返回与其它对象比较的结果 |
| 输入/输出 | |
| virtual Save | 保存对象至文件 |
| virtual Load | 从文件里读取对象 |
| virtual Type | 返回对象类型 |

© 2000-2025, MetaQuotes Ltd.

# Prev

获取列表中前一个元素的指针。

```
CObject*  Prev()
```

返回值

列表中前一个元素的指针。如果此为首个列表项，则返回 NULL。

例如：

```
//--- 例程 CObject::Prev()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("对象创建错误");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("对象创建错误");
      delete object_first;
      return;
     }
   //--- 设置关联
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- 使用前一个对象
   CObject *object=object_second.Prev();
   //--- 删除对象
   delete object_first;
   delete object_second;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Prev

将指针设置到前一个列表项。

```
void  Prev(
   CObject*  object      // 指针到前一个列表项
   )
```

参数

*object*

[输入] 前一个列表项的新指针值。

例如：

```
//--- 例程 CObject::Prev(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("对象创建错误");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("对象创建错误");
      delete object_first;
      return;
     }
   //--- 设置关联
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- 使用对象
   //--- ...
   //--- 删除对象
   delete object_first;
   delete object_second;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Next

获取列表中下一个元素的指针。

```
CObject*  Next()
```

返回值

列表中下一个元素的指针。如果此为最后的列表项，则返回 NULL。

例如：

```
//--- 例程 CObject::Next()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("对象创建错误");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("对象创建错误");
      delete object_first;
      return;
     }
   //--- 设置关联
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- 使用下一个对象
   CObject *object=object_first.Next();
   //--- 删除对象
   delete object_first;
   delete object_second;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Next

将指针设置到列表中下一个元素。

```
void  Next(
   CObject*  object      // 指针设置到下一个列表项
   )
```

参数

*object*

[输入] 下一个列表项的新指针值。

例如：

```
//--- 例程 CObject::Next(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("对象创建错误");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("对象创建错误");
      delete object_first;
      return;
     }
   //--- 设置关联
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- 使用对象
   //--- ...
   //--- 删除对象
   delete object_first;
   delete object_second;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Compare

列表中的数据项与列表中其它元素数据进行比较。

```
virtual int  Compare(
   const CObject*  node,      // 将要比较的节点
   const int       mode=0     // 比较模式
   ) const
```

参数

*node*

[输入] 进行比较的列表项指针

*mode=0*

[输入] 比较的变形

返回值

0 - 列表项相等的情况，-1 - 如果列表项小于表中另一比较的项（节点），1 - 如果列表项大于表中另一比较的项（节点）。

注释

类CObject中的方法Compare()返回0且不执行任何动作。如果您期望比较派生类的数据，则方法Compare(...)应予以实现。模式（mode）参数应在实现多元比较时使用。

例如：

```
//--- 例程 CObject::Compare(...)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("对象创建错误");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("对象创建错误");
      delete object_first;
      return;
     }
   //--- 设置关联
   object_first.Next(object_second);
   object_second.Prev(object_first);
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 比较对象
   int result=object_first.Compare(object_second);
   //--- 删除对象
   delete object_first;
   delete object_second;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数据元素列表至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄

返回值

true - 如果成功完成，false - 如果出错。

注释

类CObject中的方法 Save (int) 总是返回 true 且不执行任何动作。如果您期望派生类可保存数据至文件，则方法 Save (int) 应予以实现。

例如：

```
//--- 例程 CObject::Save(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 设置对象数据
   //--- . . .
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
```

© 2000-2025, MetaQuotes Ltd.

```
      FileClose(file_handle);
     }
   delete object;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数据项至列表。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄

返回值

true - 如果成功完成，false - 如果有错。

注释

类CObject中的方法 Load (int) 总是返回 true 且不执行任何动作。如果您期望派生类可从文件里加载数据，则方法 Load (int) 应予以实现。

例如：

```
//--- 例程 CObject::Load(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用对象
   //--- . . .
   delete object;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取类型标识符。

```
virtual int  Type() const
```

返回值

类型标识符（对应 CObject－0）。

例如：

```
//--- 例程 CObject::Type()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object=new CObject;
   //---
   object=new CObject;
   if(object ==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取对象类型
   int type=object.Type();
   //--- 删除对象
   delete object;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 数据结构

这章节包含操纵不同数据结构（数组，链表，等等）的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建不同格式的自定义数据存储（包括合成数据结构）时，使用这些数据结构类，将节省大量时间。

MQL5 标准库（就数据集而言）位于终端工作目录的 Include\Arrays 文件夹。

## 数据数组

在创建不同格式的自定义数据存储（包括多维数组）时，使用这些动态数据数组类，将节省大量时间。

MQL5 标准库（就数据数组而言）位于终端工作目录的 Include\Arrays 文件夹。

| 类 | 描述 |
| --- | --- |
| CArray | 动态数据数组基准类 |
| CArrayChar | 字符或无符号字符类型的变量动态数组 |
| CArrayShort | 短整数或无符号短整数类型的变量动态数组 |
| CArrayInt | 整数或无符号整数类型的变量动态数组 |
| CArrayLong | 长整数或无符号长整数类型的变量动态数组 |
| CArrayFloat | 浮点类型的变量动态数组 |
| CArrayDouble | 双精度类型的变量动态数组 |
| CArrayString | 字符串类型的变量动态数组 |
| CArrayObj | 动态 CObject 指针数组 |
| CList | 可供操作 CObject 及其子类的列表实例 |
| CTreeNode | 可供操作 CTree 的二叉树节点 |
| CTree | 可供操作 CTreeNode 类及其子类的二叉树实例 |

© 2000-2025, MetaQuotes Ltd.

# CArray

CArray 类是变量动态数组的基准类。

## 描述

类 CArray 旨在操作变量动态数组：内存分配，排序，以及文件操作。

## 声明

```
class CArray : public CObject
```

## 标称库文件

```
#include <Arrays\Array.mqh>
```

## 继承体系

CObject

CArray

直接分支

CArrayChar, CArrayDouble, CArrayFloat, CArrayInt, CArrayLong, CArrayObj, CArrayShort, CArrayString

## 类方法

| 属性 | |
|---|---|
| Step | 获取数组递增大小的步长 |
| Step | 设置数组递增大小 |
| Total | 获取数组中元素的数量 |
| Available | 获取数组中无需额外内存分配即可利用的自由元素数量 |
| Max | 获取数组中无需额外内存分配的最大可能尺度 |
| IsSorted | 获取数组按指定选项排序的标志 |
| SortMode | 获取数组排序的版本 |
| 清除方法 | |
| Clear | 删除数组内所有元素而无需释放内存 |
| 排序方法 | |
| Sort | 按指定选项进行数组排序 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

© 2000-2025, MetaQuotes Ltd.

## Step

获取数组大小递增的增量。

```
int  Step() const
```

返回值

数组递增大小.

例如：

```
//--- 例程 CArray::Step()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取调整步长
   int step=array.Step();
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Step

设置数组递增大小。

```
bool  Step(
   int  step      // 步长
   )
```

参数

*step*

[输入] 数组递增大小的新步长值。

返回值

true 如果成功，false - 如果尝试使用小于或等于零的步长。

例如：

```
//--- 例程 CArray::Step(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 设置调整步长
   bool result=array.Step(1024);
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Total

获取数组中元素的数量。

```
int  Total() const;
```

返回值

数组中元素的数量。

例如：

```
//--- 例程 CArray::Total()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查总数
   int total=array.Total();
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Available

获取数组中无需额外内存分配即可利用的自由元素数量。

```
int  Available() const
```

返回值

数组中无需额外内存分配即可利用的自由元素数量。

例如：

```
//--- 例程 CArray::Available()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查可用
   int available=array.Available();
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Max

获取数组中无需额外内存分配的最大可能尺度。

```
int  Max() const
```

返回值

数组中无需额外内存分配的最大可能尺度。

例如：

```
//--- 例程 CArray::Max()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查最大尺度
   int max=array.Max();
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# IsSorted

获取数组按指定选项排序的标志。

```
bool  IsSorted(
   int  mode=0      // 排序模式
   ) const
```

参数

*mode=0*

[输入] 测试排序版本。

返回值

列表排序标志。如果列表已按指定版本排序 - true，否则 - false。

注释

数组已排序征状不可直接修改。除 InsertSort(...) 之外，已排序征状可由方法 Sort() 设置，以及所有添加 / 插入方法复位。

例如：

```
//--- 例程 CArray::IsSorted()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查排序
   if(array.IsSorted())
     {
      //--- 使用已排序数组的方法
      //--- ...
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SortMode

获取数组排序的版本。

```
int  SortMode() const;
```

返回值

排序模式。

例如：

```
//--- 例程 CArray::SortMode()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查排序模式
   int sort_mode=array.SortMode();
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Clear

删除数组内所有元素而无需释放内存。

```
void  Clear()
```

返回值

无。

例如：

```
//--- 例程 CArray::Clear()
#include <Arrays\Array.mqh>
//---
void OnStart()
 {
 CArray *array=new CArray;
 //---
 if(array==NULL)
 {
 printf("对象创建错误");
 return;
 }
 //--- 使用数组
 //--- ...
 //--- 清除数组
 array.Clear();
 //--- 删除数组
 delete array;
 }
```

© 2000-2025, MetaQuotes Ltd.

# Sort

按指定选项进行数组排序。

```
void  Sort(
   int  mode=0      // 排序模式
   )
```

参数

*mode=0*

[输入] 数组排序模式。

返回值

无。

注释

数组排序永远为升序。对于原初数据类型 (CArrayChar, CArrayShort, 等等), 参数模式无效。对于数组 CArrayObj, 多元排序应当在派生类的 Sort (int) 方法里实现。

例如：

```
//--- 例程 CArray::Sort(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 排序模式 0
   array.Sort(0);
   //--- 使用数组
   //--- ...
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArray::Save(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArray::Load(...)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayChar

CArrayChar 类是字符或无符号字符类型的变量动态数组。

## 描述

类 CArrayChar 可供操作字符或无符号字符类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayChar : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayChar.mqh>
```

## 继承体系

CObject
  CArray
    CArrayChar

## 类方法

| 内存控制 | |
|---|---|
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 修改方法 | |
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 内存控制 | |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 数组排序方法 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与样本相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayChar::Reserve(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayChar::Resize(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayChar::Shutdown()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   char  element      // 添加元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayChar::Add(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const char&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayChar::AddArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayChar*  src      // 源数组指针
   )
```

参数

*src*

[输入] CArrayChar类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayChar::AddArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   char  element,     // 插入元素
   int   pos          // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayChar::Insert(char,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const char&  src[],     // 源数组
   int           pos       // 位置
   )
```

参数

*src[]*

[输入] 源元素的数组引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayChar::InsertArray(const char &[],int)
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayChar*  src,      // 源指针
   int           pos      // 位置
   )
```

参数

*src*

[输入] CArrayChar类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayChar::InsertArray(const CArrayChar*,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const char&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayChar::AssignArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayChar*  src      // 源数组指针
   )
```

参数

*src*

[输入] CArrayChar类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayChar::AssignArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayChar *src  =new CArrayChar;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int   pos,          // 位置
   char  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayChar::Update(int,char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,'A'))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 值
   )
```

参数

*pos*

[输入] 在数组中移动元素位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayChar::Shift(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayChar::Delete(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayChar::DeleteRange(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
char  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，CHAR_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，CHAR_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayChar::At(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      char result=array.At(i);
      if(result==CHAR_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const char&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayChar::CompareArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const CArrayChar*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayChar类实例的指针-用于比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayChar::CompareArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   char  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayChar::InsertSort(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort('A'))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::Search(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search('A')!=-1) printf("元素发现");
   else                      printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchGreat(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat('A')!=-1) printf("元素发现");
   else                           printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchLess(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess('A')!=-1) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素

```
int  SearchGreatOrEqual(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchGreatOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual('A')!=-1) printf("元素发现");
   else                                  printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchLessOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual('A')!=-1) printf("元素发现");
   else                                 printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchFirst(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst('A')!=-1) printf("元素发现");
   else                           printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayChar::SearchLast(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast('A')!=-1) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   char  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayChar::SearchLinear(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear('A')!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果有错。

例如：

```
//--- 例程 CArrayChar::Save(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果有错。

例如：

```
//--- 例程 CArrayChar::Load(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = '%c'",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayChar - 77)。

例如：

```
//--- 例程 CArrayChar::Type()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayShort

CArrayShort 类是短整数或无符号短整数类型的变量动态数组。

## 描述

类 CArrayShort 可供操作短整数或无符号短整数类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayShort : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayShort.mqh>
```

## 继承体系

CObject
  CArray
    CArrayShort

## 类方法

| 内存控制 | |
|---|---|
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 内存控制 | |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayShort::Reserve(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayShort::Resize(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayShort::Shutdown()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   short  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayShort::Add(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const short&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayShort::AddArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayShort*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayShort 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayShort::AddArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   short  element,      // 插入的元素
   int    pos           // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayShort::Insert(short,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const short&  src[],     // 源数组
   int           pos        // 位置
   )
```

参数

*src[]*

[输入] 源元素的数组引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayShort::InsertArray(const short &[],int)
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayShort*  src,      // 源指针
   int            pos      // 位置
   )
```

参数

*src*

[输入] CArrayShort 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayShort::InsertArray(const CArrayShort*,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const short&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayShort::AssignArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayShort*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayShort 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayShort::AssignArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayShort *src  =new CArrayShort;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int    pos,          // 位置
   short  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayShort::Update(int,short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,100))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayShort::Shift(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayShort::Delete(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayShort::DeleteRange(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
short  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，SHORT_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，SHORT_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayShort::At(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      short result=array.At(i);
      if(result==SHORT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const short&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayShort::CompareArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const CArrayShort*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayShort 类实例的指针-用于比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayShort::CompareArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   short  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayShort::InsertSort(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(100))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::Search(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search(100)!=-1) printf("元素发现");
   else                      printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchGreat(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat(100)!=-1) printf("元素发现");
   else                           printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchLess(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess(100)!=-1) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchGreatOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual(100)!=-1) printf("元素发现");
   else                                  printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchLessOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual(100)!=-1) printf("元素发现");
   else                                 printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchFirst(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst(100)!=-1) printf("元素发现");
   else                           printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayShort::SearchLast(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast(100)!=-1) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   short  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayShort::SearchLinear(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear(100)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayShort::Save(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayShort::Load(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = %d",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayShort - 82)。

例如：

```
//--- 例程 CArrayShort::Type()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayInt

CArrayInt 类是整数或无符号整数类型的变量动态数组。

## 描述

类 CArrayInt 可供操作整数或无符号整数类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayInt : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayInt.mqh>
```

## 继承体系

CObject
  CArray
    CArrayInt

直接分支

CSpreadBuffer

## 类方法

| 内存控制 | |
|---|---|
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |

© 2000-2025, MetaQuotes Ltd.

| 内存控制 | |
|---|---|
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayInt::Reserve(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayInt::Resize(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayInt::Shutdown()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   int  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayInt::Add(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const int&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayInt::AddArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayInt*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayInt 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayInt::AddArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   int  element,     // 插入的元素
   int  pos          // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayInt::Insert(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const int&  src[],     // 源数组
   int         pos        // 位置
   )
```

参数

*src[]*

[输入] 源元素的数组引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayInt::InsertArray(const int &[],int)
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayInt*  src,      // 源指针
   int          pos      // 位置
   )
```

参数

*src*

[输入] CArrayInt 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayInt::InsertArray(const CArrayInt*,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const int&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayInt::AssignArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayInt*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayInt 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayInt::AssignArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayInt *src  =new CArrayInt;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 数组已标识
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int  pos,          // 位置
   int  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayInt::Update(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,10000))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayInt::Shift(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayInt::Delete(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayInt::DeleteRange(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
int  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，INT_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，INT_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayInt::At(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      int result=array.At(i);
      if(result==INT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
   //--- 删除数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const int&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayInt::CompareArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const CArrayInt*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayInt 类实例的指针-用于比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayInt::CompareArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   int  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayInt::InsertSort(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(10000))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::Search(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search(10000)!=-1) printf("元素发现");
   else                        printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::SearchGreat(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat(10000)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::SearchLess(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess(10000)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   int  element      // 搜索的元素
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::SearchGreatOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual(10000)!=-1) printf("元素发现");
   else                                    printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::SearchLessOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual(10000)!=-1) printf("元素发现");
   else                                   printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt:: SearchFirst(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst(10000)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayInt::SearchLast(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast(10000)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   int  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayInt::SearchLinear(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear(10000)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayInt::Save(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayInt::Load(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = %d",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayInt - 82)。

例如：

```
//--- 例程 CArrayInt::Type()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayLong

CArrayLong 类是长整数或无符号长整数类型的变量动态数组。

## 描述

类 CArrayLong 可供操作长整数或无符号长整数类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayLong : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayLong.mqh>
```

## 继承体系

CObject
  CArray
    CArrayLong

直接分支

CRealVolumeBuffer, CTickVolumeBuffer, CTimeBuffer

## 类方法

| 内存控制 | |
|---|---|
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |

© 2000-2025, MetaQuotes Ltd.

| 内存控制 | |
|---|---|
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayLong::Reserve(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayLong::Resize(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayLong::Shutdown()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   long  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayLong::Add(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const long&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayLong::AddArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayLong*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayLong 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayLong::AddArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   long  element,     // 插入的元素
   int   pos          // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayLong::Insert(long,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const long&  src[],     // 源数组
   int           pos        // 位置
   )
```

参数

*src[]*

[输入] 源元素的数组引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayLong::InsertArray(const long &[],int)
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayLong*  src,      // 源指针
   int           pos      // 位置
   )
```

参数

*src*

[输入] CArrayLong 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayLong::InsertArray(const CArrayLong*,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const long&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayLong::AssignArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayLong*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayLong 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayLong::AssignArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayLong *src  =new CArrayLong;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int   pos,          // 位置
   long  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayLong::Update(int,long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,1000000))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayLong::Shift(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayLong::Delete(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayLong::DeleteRange(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
long  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，LONG_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，LONG_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayLong::At(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      long result=array.At(i);
      if(result==LONG_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const long&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayLong::CompareArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArrayconst

比较数组与另一个数组。

```
bool  CompareArrayconst(
   const CArrayLong*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayLong 类实例的指针-用于比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayLong::CompareArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   long  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayLong::InsertSort(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(1000000))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与模型相等的元素

```
int  Search(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::Search(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search(1000000)!=-1) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索比样本更大的元素。

```
int  SearchGreat(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchGreat(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat(1000000)!=-1) printf("元素发现");
   else                               printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchLess(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess(1000000)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于模型的元素。

```
int  SearchGreatOrEqual(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchGreatOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual(1000000)!=-1) printf("元素发现");
   else                                      printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于模型的元素。

```
int  SearchLessOrEqual(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchLessOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual(1000000)!=-1) printf("元素发现");
   else                                     printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与模型相等的元素。

```
int  SearchFirst(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchFirst(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst(1000000)!=-1) printf("元素发现");
   else                               printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayLong::SearchLast(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast(1000000)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   long  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayLong::SearchLinear(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear(1000000)!=-1) printf("元素发现");
   else                                printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果有错。

例如：

```
//--- 例程 CArrayLong::Save(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果有错。

例如：

```
//--- 例程 CArrayLong::Load(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = %I64",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayLong - 84)。

例如：

```
//--- 例程 CArrayLong::Type()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayFloat

CArrayFloat 类是浮点类型的变量动态数组。

## 描述

类 CArrayFloat 可供操作浮点类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayFloat : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayFloat.mqh>
```

## 继承体系

CObject
  CArray
    CArrayFloat

## 类方法

| 属性 | |
|---|---|
| Delta | 设置比较冗余度 |
| 内存控制 | |
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Delta

设置比较冗余度。

```
void  Delta(
   float  delta      // 冗余度
   )
```

参数

*delta*

[输入] 比较时可接纳的新公差值。

返回值

None

注释

在搜索时进行比较的可接纳值。如果它们的差别小于或等于冗余度，则该值被认为是相等的。省缺冗余度是 0.0。

例如：

```
//--- 例程 CArrayFloat::Delta(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 设置比较公差
   array.Delta(0.001);
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayFloat::Reserve(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true 如果成功，false - 如果尝试设置小于或等于零的大小。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayFloat::Resize(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayFloat::Shutdown()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   float  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayFloat::Add(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const float&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayFloat::AddArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayFloat*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayFloat 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayFloat::AddArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   float  element,      // 插入的元素
   int    pos           // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayFloat::Insert(float,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const float&  src[],     // 源数组
   int           pos        // 位置
   )
```

参数

*src[]*

[输入] 源元素的数组引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayFloat::InsertArray(const float &[],int)
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayFloat*  src,      // 源指针
   int            pos      // 位置
   )
```

参数

*src*

[输入] CArrayFloat 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayFloat::InsertArray(const CArrayFloat*,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const float&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayFloat::AssignArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayFloat*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayFloat 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayFloat::AssignArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayFloat *src  =new CArrayFloat;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int    pos,          // 位置
   float  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayFloat::Update(int,float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,100.0))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功, false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayFloat::Shift(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayFloat::Delete(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayFloat::DeleteRange(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
float  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，FLT_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，FLT_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayFloat::At(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      float result=array.At(i);
      if(result==FLT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
   //--- 删除数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const float&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayFloat::CompareArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArrayconst

从另一个数组里拷贝元素至本数组。

```
bool  AssignArrayconst(
   const CArrayFloat*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayFloat 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayFloat::CompareArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   float  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayFloat::InsertSort(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(100.0))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::Search(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search(100.0)!=-1) printf("元素发现");
   else                        printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::SearchGreat(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat(100.0)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat:: SearchLess(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess(100.0)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::SearchGreatOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("元素发现");
   else                                    printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::SearchLessOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual(100.0)!=-1) printf("元素发现");
   else                                   printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::SearchFirst(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst(100.0)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayFloat::SearchLast(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast(100.0)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   float  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayFloat::SearchLinear(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear(100.0)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayFloat::Save(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayFloat::Load(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = %f",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayFloat - 87)。

例如：

```
//--- 例程 CArrayFloat::Type()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayDouble

CArrayDouble 类是双精度类型的变量动态数组。

## 描述

类 CArrayDouble 可供操作双精度类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayDouble : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayDouble.mqh>
```

## 继承体系

CObject
  CArray
    CArrayDouble

直接分支

CDoubleBuffer

## 类方法

| 属性 | |
|---|---|
| Delta | 设置比较冗余度 |
| 内存控制 | |
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| 搜索最小/最大 | |
| Minimum | 获取数组中指定范围内最小元素的索引 |
| Maximum | 获取数组中指定范围内最大元素的索引 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Delta

设置比较冗余度。

```
void  Delta(
   double  delta      // 冗余度
   )
```

参数

*delta*

[输入] 比较时可接纳的新公差值。

返回值

无

注释

在搜索时进行比较的可接纳值。如果它们的差别小于或等于冗余度，则该值被认为是相等的。省缺冗余度是 0.0。

例如：

```
//--- 例程 CArrayDouble::Delta(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 设置比较公差
   array.Delta(0.001);
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayDouble::Reserve(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayDouble::Resize(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayDouble::Shutdown()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   double  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayDouble::Add(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const double&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayDouble::AddArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayDouble*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayDouble 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayDouble::AddArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   double  element,     // 插入的元素
   int     pos          // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayDouble::Insert(double,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const double&  src[],      // 源数组
   int            pos         // 位置
   )
```

参数

*src[]*

[输入] 插入源元素的数组的引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayDouble::InsertArray(const double &[],int)
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayDouble*  src,      // 源指针
   int            pos       // 位置
   )
```

参数

*src*

[输入] CArrayDouble 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayDouble::InsertArray(const CArrayDouble*,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const double&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayDouble::AssignArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayDouble*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayDouble 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayDouble::AssignArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayDouble *src  =new CArrayDouble;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int     pos,          // 位置
   double  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayDouble::Update(int,double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,100.0))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素的位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayDouble::Shift(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayDouble::Delete(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayDouble::DeleteRange(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
double  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，DBL_MAX-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，DBL_MAX 可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayDouble::At(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      double result=array.At(i);
      if(result==DBL_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //---  读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
   //--- 删除数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const double&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源元素数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayDouble::CompareArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const CArrayDouble*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayDouble 类实例的指针-用于比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayDouble::CompareArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Minimum

获取数组中指定范围内最小元素的索引。

```
int  Minimum(
   int  start,      // 起始索引
   int  count       // 处理元素的数量
   ) const
```

参数

*start*

[输入] 数组的起始索引。

*count*

[输入] 处理元素的数量。

返回值

数组中指定范围内最小元素的索引。

© 2000-2025, MetaQuotes Ltd.

# Maximum

获取数组中指定范围内最大元素的索引。

```
int  Maximum(
   int  start,      // 起始索引
   int  count       // 处理元素的数量
   ) const
```

参数

*start*

[输入] 数组的起始索引。

*count*

[输入] 处理元素的数量。

返回值

数组中指定范围内最大元素的索引。

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   double  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayDouble::InsertSort(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(100.0))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::Search(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search(100.0)!=-1) printf("元素发现");
   else                        printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::SearchGreat(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat(100.0)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble:: SearchLess(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess(100.0)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::SearchGreatOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("元素发现");
   else                                    printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::SearchLessOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual(100.0)!=-1) printf("元素发现");
   else                                   printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::SearchFirst(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst(100.0)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayDouble::SearchLast(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast(100.0)!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在排序数组里搜索与样本相等的元素。

```
int  SearchLinear(
   double  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayDouble::SearchLinear(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear(100.0)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayDouble::Save(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int            file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayDouble::Load(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int            file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = %f",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayDouble - 87)。

例如：

```
//--- 例程 CArrayDouble::Type()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayString

CArrayString 类是字符串类型的变量动态数组。

## 描述

类 CArrayString 可供操作字符串类型的变量动态数组。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。此外，还实现了文件操作方法。

## 声明

```
class CArrayString : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayString.mqh>
```

## 继承体系

CObject
  CArray
    CArrayString

## 类方法

| 内存控制 | |
|---|---|
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 设置数组新大小（更小） |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| AddArray | 在数组末尾添加另一个数组的元素 |
| Insert | 在数组中按指定位置插入元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 内存控制 | |
| 删除方法 | |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| CompareArray | 比较数组与另一个数组 |
| Sorted array opetations | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与样本相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于样本的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于样本的元素 |
| SearchFirst | 在排序数组里搜索首个与样本相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与样本相等的元素 |
| SearchLinear | 在数组里搜索与样本相等的元素 |
| 输入/输出 | |
| virtual Save | 保存数组数据至文件 |
| virtual Load | 从文件里加载数组数据。 |
| virtual Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayString::Reserve(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 保留内存
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。右侧的多余元素丢失。为降低内存碎片， 增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayString::Resize(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

例如：

```
//--- 例程 CArrayString::Shutdown()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   string  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

例如：

```
//--- 例程 CArrayString::Add(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(IntegerToString(i)))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const string&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 添加元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayString::AddArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayString*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayString 类实例的指针-添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

例如：

```
//--- 例程 CArrayString::AddArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 添加另一个数组
   if(!array.AddArray(src))
     {
      printf("数组添加错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除源数组
   delete src;
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   string  element,     // 插入的元素
   int     pos          // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayString::Insert(string,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(IntegerToString(i),0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const string&  src[],     // 源数组
   int            pos        // 位置
   )
```

参数

*src[]*

[输入] 插入源元素的数组的引用。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayString::InsertArray(const string &[],int)
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   CArrayString*  src,      // 源指针
   int             pos      // 位置
   )
```

参数

*src*

[输入] CArrayString 类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

例如：

```
//--- 例程 CArrayString::InsertArray(const CArrayString*,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 插入其它数组
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
   //--- 删除源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const string&  src[]      // 源数组
   )
```

参数

*src[]*

[输入] 被拷贝元素的源数组的引用。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayString::AssignArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayString*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayString 类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

例如：

```
//--- 例程 CArrayString::AssignArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayString *src  =new CArrayString;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除源数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int     pos,          // 位置
   string  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

例如：

```
//--- 例程 CArrayString::Update(int, string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,"ABC"))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素的位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayString::Shift(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayString::Delete(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

例如：

```
//--- 例程 CArrayString::DeleteRange(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
string  At(
   int  pos      // 位置
   ) const
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，"-如果尝试获取位置不存在的元素 (最后的错误 ERR_OUT_OF_RANGE)。

注释

当然，"可能是一个数组元素的有效值，因此若有返回值，永远检查最后的错误代码。

例如：

```
//--- 例程 CArrayString::At(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      string result=array.At(i);
      if(result=="" && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
   //--- 删除数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const string&  src[]      // 源数组
   ) const
```

参数

*src[]*

[输入] 用于比较的源元素数组的引用。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayString::CompareArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArrays(
   const CArrayString*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayString 类实例的指针-比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayString::CompareArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 添加源数组元素
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   string  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

例如：

```
//--- 例程 CArrayString::InsertSort(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort("ABC"))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString::Search(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.Search("ABC")!=-1) printf("元素发现");
   else                        printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString::SearchGreat(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreat("ABC")!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 元素发现 CArrayString:: SearchLess(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLess("ABC")!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString:: SearchGreatOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchGreatOrEqual("ABC")!=-1) printf("元素发现");
   else                                    printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString:: SearchLessOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLessOrEqual("ABC")!=-1) printf("元素发现");
   else                                   printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString:: SearchFirst(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchFirst("ABC")!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayString:: SearchLast(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 搜索元素
   if(array.SearchLast("ABC")!=-1) printf("元素发现");
   else                            printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLinear

在数组里搜索与样本相等的元素。

```
int  SearchLinear(
   string  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果元素未发现。

注释

对于未排序数组，方法使用线性搜索（或顺序搜索）算法。

例如：

```
//--- 例程 CArrayString::SearchLinear(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 搜索元素
   if(array.SearchLinear("ABC")!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayString::Save(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int            file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      array.Add(IntegerToString(i));
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayString::Load(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
   for(int i=0;i<array.Total();i++)
     {
      printf("元素[%d] = '%s'",i,array.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的类型 ID (对于 CArrayString - 89)。

例如：

```
//--- 例程 CArrayString::Type()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CArrayObj

CArrayObj类是存储 CObject 及其继承类的实例指针的变量动态数组。

## 描述

类 CArrayObj 可供操作 CObject 及其继承类的实例指针的变量动态数组。它为操纵那些原初数据类型和较难组织的数据结构的多维动态数组提供了可能。

类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。. 此外，还实现了文件操作方法。

还有某些 CArrayObj的 细微之处 。

## 声明

```
class CArrayObj : public CArray
```

## 标称库文件

```
#include <Arrays\ArrayObj.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj

直接分支

CIndicators, CSeries

## 类方法

| 属性 | |
|---|---|
| FreeMode | 获取内存管理标志 |
| FreeMode | 设置内存管理标志 |
| 内存控制 | |
| Reserve | 分配内存增加数组大小 |
| 调整大小 | 设置数组新大小（更小） |
| Shutdown | 清除数组并完全释放内存数组（非元素）。 |
| 添加方法 | |
| 添加 | 在数组末尾添加元素 |
| AddArray | 在数组末尾添加元素 |
| Insert | 在数组中按指定位置插入元素 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| InsertArray | 在数组中按指定位置插入另一个数组的元素 |
| AssignArray | 从另一个数组里拷贝元素至本数组 |
| 更新方法 | |
| Update | 在指定数组位置修改元素 |
| 平移 | 在数组里从给定位置移动一个元素至另一个指定偏移。 |
| 删除方法 | |
| Detach | 从数组指定位置获取元素，并将其删除。 |
| 删除 | 从数组的指定位置删除元素 |
| DeleteRange | 从数组的指定位置删除一组元素 |
| Clear | 删除数组内所有元素而无需释放内存数组 |
| 访问方法 | |
| At | 获取数组指定位置的元素 |
| 比较方法 | |
| CompareArray | 比较数组与另一个数组 |
| 数组排序操作 | |
| InsertSort | 在排序数组里插入元素 |
| 搜索 | 在排序数组里搜索与模型相等的元素 |
| SearchGreat | 在排序数组里搜索更多与样本相等的元素 |
| SearchLess | 在排序数组里搜索小于样本的元素 |
| SearchGreatOrEqual | 在排序数组里搜索大于或等于模型的元素 |
| SearchLessOrEqual | 在排序数组里搜索小于或等于模型的元素 |
| SearchFirst | 在排序数组里搜索首个与模型相等的元素 |
| SearchLast | 在排序数组里搜索最后一个与模型相等的元素 |
| 输入/输出 | |
| Save | 保存数组数据至文件 |
| Load | 从文件里加载数组数据。 |
| Type | 获取数组类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

数组的实际应用是CObject类的子类(包括标准库里的所有类)。

例如,考虑二维数组作为选项:

```
#include <Arrays\ArrayDouble.mqh>
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int i,j;
   int first_size=10;
   int second_size=100;
//--- 创建数组
   CArrayObj    *array=new CArrayObj;
   CArrayDouble *sub_array;
//---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
//--- 创建子数组
   for(i=0;i<first_size;i++)
     {
      sub_array=new CArrayDouble;
      if(sub_array==NULL)
        {
         delete array;
         printf("对象创建错误");
         return;
        }
      //--- 填充数组
      for(j=0;j<second_size;j++)
        {
         sub_array.Add(i*j);
        }
      array.Add(sub_array);
     }
//--- 创建数组成功
   for(i=0;i<first_size;i++)
     {
      sub_array=array.At(i);
      for(j=0;j<second_size;j++)
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         double element=sub_array.At(j);
         //--- 使用数组元素
        }
     }
   delete array;
  }
```

## 细节

这个类具有控制动态内存机制，所以操作数组中的元素时要小心。

内存管理机制可以使用方法 FreeMode (bool) 切换。省缺时，机制启用。

因此，有两种选择来处理类CArrayObj:

1. 内存管理机制启用。(省缺)

此种情况，当元素从数组删除后 CArrayObj负责释放其占用内存。在此程序里，用户不应释放数组元素。

例如：

```
   int i;
//--- 创建数组
   CArrayObj *array=new CArrayObj;
//--- 填充数组元素
   for(i=0;i<10;i++) array.Add(new CObject);
//--- 做点什么
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- 利用元素的动作
      . . .
     }
//--- 删除数组元素
   delete array;
```

2. 内存管理机制关闭。

此种情况，当元素从数组删除后 CArrayObj不负责释放其占用内存。在此程序里，用户必须释放数组元素。

例如：

```
   int i;
//--- 创建数组
   CArrayObj *array=new CArrayObj;
//--- 禁用内存管理机制
   array.FreeMode(false);
//--- 填充数组元素
   for(i=0;i<10;i++) array.Add(new CObject);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 做点什么
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- 利用元素的动作
      . . .
     }
//--- 删除数组元素
   while(array.Total()) delete array.Detach();
//--- 删除空数组
   delete array;
```

© 2000-2025, MetaQuotes Ltd.

# FreeMode

获取内存管理标志。

```
bool  FreeMode() const
```

返回值

内存管理标志。

例如：

```
//--- 例程 CArrayObj::FreeMode()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取释放模式标志
   bool array_free_mode=array.FreeMode();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# FreeMode

设置内存管理标志。

```
void  FreeMode(
   bool  mode      // 新标志
   )
```

参数

*mode*

[输入] 新的内存管理标志值。

返回值

无。

注释

设置内存管理标志 - 这是使用类CArrayObj的重点。由于数组元素是动态对象指针，当从数组删除时判断做什么就很重要。

如果标志已设置，从数组删除元素时元素自动由操作符删除。如果标志未设置，则假设删除对象的指针在用户程序的某处依然使用，且程序随后会将其释放。

如果用户重置了内存管理标志，用户必须了解他们有责任在程序完成前删除数组，因为不这样做的话，在创建新的操作符时原有元素所占用的内存不会被释放。

当数据量很大时，它会产生泄露，最终甚至会破坏您的终端。如果用户没有重置内存管理标志，还有另一个"暗礁"。

使用存储在本地变量某处的数组元素指针，  当删除数组后将会导致严重错误并令用户程序崩溃。省缺时，内存管理标志被设置，即数组类负责释放内存元素。

例如：

```
//--- 例程 CArrayObj::FreeMode(bool)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 重置模式标志
   array.FreeMode(false);
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# Reserve

分配内存增加数组大小。

```
bool  Reserve(
   int  size      // 数量
   )
```

参数

*size*

[输入] 数组附加元素的数量。

返回值

true 如果成功，false - 如果尝试谋求小于或等于零的容量，或数组未能增加。

注释

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayObj::Reserve(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   if(!array.Reserve(1024))
     {
      printf("保留错误");
      delete array;
      return;
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置数组新大小（更小）。

```
bool  Resize(
   int  size      // 大小
   )
```

参数

*size*

[输入] 数组的新大小。

返回值

true - 成功，false - 试图将其大小设置为小于零。

注释

改变数组大小可以优化内存使用。位于右侧的超出元素都将丢失。丢失元素的内存被释放，或是不依据存储器管理的模式。

为降低内存碎片，增加数组大小由之前通过 Step (int) 方法给定的步长，或 16 (省缺) 来进行。

例如：

```
//--- 例程 CArrayObj::Resize(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 调整数组大小
   if(!array.Resize(10))
     {
      printf("调整大小错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Clear

删除数组内所有元素而无需释放内存数组。

```
void  Clear()
```

返回值

无。

注释

如果启用内存管理机制，删除的元素被释放。

例如：

```
//--- 例程 CArrayObj::Clear()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 清除数组
   array.Clear();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Shutdown

清除数组并完全释放内存数组（非元素）。

```
bool  Shutdown()
```

返回值

true 如果成功，false 如果错误发生。

注释

如果启用内存管理机制，删除的元素被释放。

例如：

```
//--- 例程 CArrayObj::Shutdown()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 关闭数组
   if(!array.Shutdown())
     {
      printf("关闭错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CreateElement

在指定数组位置创建新元素。

```
bool  CreateElement(
   int  index      // 位置
   )
```

参数

*index*

[输入] 您期望创建新元素的位置。

返回值

true 如果成功，false - 如果您未能创建元素。

注释

在 CArrayObj 类中的方法 CreateElement (int) 永远返回 false 且不执行任何动作。如果必要，在派生类中，方法 CreateElement (int) 应予实现。

例如：

```
//--- 例程 CArrayObj::CreateElement(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int         size=100;
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 填充数组
   array.Reserve(size);
   for(int i=0;i<size;i++)
     {
      if(!array.CreateElement(i))
        {
         printf("元素创建错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在数组末尾添加元素。

```
bool  Add(
   CObject*  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到数组的元素值。

返回值

true 如果成功，false - 如果您未能添加元素。

注释

如果传递无效指针值 (比如 NULL)，则元素不会被加入数组。

例如：

```
//--- 例程 CArrayObj::Add(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个 数组元素
   for(int i=0;i<100;i++)
     {
      if(!array.Add(new CObject))
        {
         printf("元素添加错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AddArray

在数组末尾添加另一个数组的元素。

```
bool  AddArray(
   const CArrayObj *  src      // 源 数组指针
   )
```

参数

*src*

[输入] 类 CArrayDouble 的实例指针 - 添加的源元素。

返回值

true 如果成功，false - 如果您未能添加项目。

注释

在数组里添加元素实际上是拷贝指针。所以，当调用方法时，此处是一个 "暗礁" - 有一个指向动态对象的指针存在于多个变量里。

```
//--- 例如
extern bool       make_error;
extern int        error;
extern CArrayObj *src;
//--- 创建 CArrayObj 的新实例
//--- 省缺内存管理打开
CArrayObj *array=new CArrayObj;
//--- 添加 (拷贝) 元素至源数组
if(array!=NULL)
   bool result=array.AddArray(src);
if(make_error)
  {
   //--- 执行错误的动作
   switch(error)
     {
      case 0:
         //--- 不检查它的内存管理标志, 直接删除源数组
         delete src;
         //--- 结果:
         //--- 在接收数组里, 有可能会以无效指针来定位元素
         break;
      case 1:
         //--- 在源数组里禁用内存管理机制
         if(src.FreeMode()) src.FreeMode(false);
         //--- 但是不删除源数组
         //--- 结果:
         //--- 在删除接收数组后, 有可能源数组里会以无效指针来定位元素
         break;
      case 2:
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 在源数组里禁用内存管理机制
         src.FreeMode(false);
         //--- 在接收数组里禁用内存管理机制
         array.FreeMode(false);
         //--- 结果:
         //--- 在程序终止时得到一个 "内存溢出"
         break;
      }
   }
else
   {
    //--- 在源数组里禁用内存管理机制
    if(src.FreeMode()) src.FreeMode(false);
    //--- 删除源数组
    delete src;
    //--- 结果:
    //--- 定位接收数组元素正确
    //--- 删除接收数组将会导致删除其元素
   }
```

例如：

```
//--- 例程 CArrayObj::AddArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
   {
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
      {
       printf("对象创建错误");
       return;
      }
    //--- 创建源数组
    CArrayObj *src=new CArrayObj;
    if(src==NULL)
      {
       printf("对象创建错误");
       delete array;
       return;
      }
    //--- 重置模式标志
    src.FreeMode(false);
    //--- 填充源数组
    //--- . . .
    //--- 添加另一个数组
    if(!array.AddArray(src))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       printf("数组添加错误");
       delete src;
       delete array;
       return;
      }
   //--- 删除无元素源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在数组中按指定位置插入元素。

```
bool  Insert(
   CObject*  element,      // 插入的元素
   int       pos           // 位置
   )
```

参数

*element*

[输入] 插入到数组的元素值。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入元素。

注释

如果传递无效指针值 (比如 NULL)，则元素不会被加入数组。

例如：

```
//--- 例程 CArrayObj::Insert(CObject*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入元素
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(new CObject,0))
        {
         printf("插入错误");
         delete array;
         return;
        }
     }
   //--- 使用数组
   //--- . . .
   //--- 删除数组
```

© 2000-2025, MetaQuotes Ltd.

```
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertArray

在数组中按指定位置插入另一个数组的元素。

```
bool  InsertArray(
   const CArrayObj*  src,     // 源指针
   int               pos      // 位置
   )
```

参数

*src*

[输入] CArrayObj类实例的指针-插入的源元素。

*pos*

[输入] 插入到数组的位置。

返回值

true 如果成功，false - 如果您未能插入项目。

注释

参阅：CArrayObj::AddArray(const CArrayObj*)。

例如：

```
//--- 例程 CArrayObj::InsertArray(const CArrayObj*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 重置模式标志
   src.FreeMode(false);
   //--- 填充源数组
   //--- . . .
   //--- 插入其它数组
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!array.InsertArray(src,0))
     {
      printf("数组插入错误");
      delete src;
      delete array;
      return;
     }
   //--- 删除无元素源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# AssignArray

从另一个数组里拷贝元素至本数组。

```
bool  AssignArray(
   const CArrayObj*  src      // 源指针
   )
```

参数

*src*

[输入] CArrayObj类实例的指针-拷贝的源元素。

返回值

true 如果成功，false - 如果您未能拷贝项目。

注释

如果 AssignArray 反馈数组不为空，其所有元素将被从数组删除，如果内存管理机制启用，删除的项目将被释放。接收数组是源数组的精确副本。另外参阅 CArrayObj::AddArray(const CArrayObj*)。

例如：

```
//--- 例程 CArrayObj::AssignArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 重置模式标志
   src.FreeMode(false);
   //--- 填充源数组
   //--- . . .
   //--- 分配其它数组
   if(!array.AssignArray(src))
     {
```

© 2000-2025, MetaQuotes Ltd.

```
      printf("数组分配错误");
      delete src;
      delete array;
      return;
     }
   //--- 数组已标识
   //--- 删除无元素源数组
   delete src;
   //--- 使用数组
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

在指定数组位置修改元素。

```
bool  Update(
   int       pos,          // 位置
   CObject*  element       // 值
   )
```

参数

*pos*

[输入] 数组中要修改的元素的位置。

*element*

[输入] 新元素值。

返回值

true 如果成功，false - 如果您未能修改元素。

注释

如果我们传递的参数是无效指针 (即 NULL)，元素不会被修改。如果启用了内存管理，内存占位符释放。

例如：

```
//--- 例程 CArrayObj::Update(int,CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 更新元素
   if(!array.Update(0,new CObject))
     {
      printf("更新错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 平移

在数组里从给定位置移动一个元素至另一个指定偏移。

```
bool  Shift(
   int  pos,        // 位置
   int  shift       // 平移
   )
```

参数

*pos*

[输入] 在数组中移动元素的位置。

*shift*

[输入] 位移值 (正负值均可)。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CArrayObj::Shift(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 平移元素
   if(!array.Shift(10,-5))
     {
      printf("平移错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Detach

从数组里删除给定位置的元素。

```
CObject*  Detach(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 在数组中的元素位置。

返回值

成功情况下是删除元素的指针，NULL - 如果您未能删除元素。

注释

无论内存管理标志处于什么状态，从数组删除时元素并未删除。从使用后释放的成分里抽取数组元素的指针。

例如：

```
//--- 例程 CArrayObj::Detach(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   CObject *object=array.Detach(0);
   if(object==NULL)
     {
      printf("Detach error");
      delete array;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 删除元素
   delete object;
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除数组指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 删除元素在数组中的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

注释

如果启用内存管理机制，删除的元素被释放。

例如：

```
//--- 例程 CArrayObj::Delete(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   if(!array.Delete(0))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteRange

从数组的指定位置删除一组元素。

```
bool  DeleteRange(
   int  from,     // 首元素位置
   int  to        // 最后元素位置
   )
```

参数

*from*

[输入] 在数组中的首个删除元素位置。

*to*

[输入] 在数组中的最后删除元素位置。

返回值

true 如果成功，false - 如果您未能删除元素。

注释

如果启用内存管理机制，删除的元素被释放。

例如：

```
//--- 例程 CArrayObj::DeleteRange(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 删除元素
   if(!array.DeleteRange(0,10))
     {
      printf("删除错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# At

获取数组指定位置的元素。

```
CObject*  At(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 在数组中的期望元素位置。

返回值

成功情况下则为元素值，NULL-如果尝试获取位置不存在的元素。

例如：

```
//--- 例程 CArrayObj::At(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加元素
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      CObject *result=array.At(i);
      if(result==NULL)
        {
         //--- 读数组错误
         printf("获取元素错误");
         delete array;
         return;
        }
      //--- 使用元素
      //--- . . .
     }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareArray

比较数组与另一个数组。

```
bool  CompareArray(
   const CArrayObj*  src      // 源指针
   ) const
```

参数

*src*

[输入] CArrayObj类实例的指针-比较的源元素。

返回值

true 如果数组相等，false - 如果不等。

例如：

```
//--- 例程 CArrayObj::CompareArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源数组
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete array;
      return;
     }
   //--- 填充源数组
   //--- . . .
   //--- 与另一数组比较
   int result=array.CompareArray(src);
   //--- 删除数组
   delete src;
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# InsertSort

在排序数组里插入元素。

```
bool  InsertSort(
   CObject*  element      // 插入的元素
   )
```

参数

*element*

[输入] 插入到排序数组的元素值。

返回值

true 如果成功，false - 如果您未能插入元素。

注释

如果传递无效指针值 (比如 NULL)，则元素不会被加入数组。

例如：

```
//--- 例程 CArrayObj::InsertSort(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 插入元素
   if(!array.InsertSort(new CObject))
     {
      printf("插入错误");
      delete array;
      return;
     }
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序数组里搜索与样本相等的元素。

```
int  Search(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj::Search(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.Search(sample)!=-1) printf("元素发现");
   else                         printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreat

在排序数组里搜索更多与样本相等的元素。

```
int  SearchGreat(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj::SearchGreat(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchGreat(sample)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLess

在排序数组里搜索小于样本的元素。

```
int  SearchLess(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj:: SearchLess(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchLess(sample)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchGreatOrEqual

在排序数组里搜索大于或等于样本的元素。

```
int  SearchGreatOrEqual(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj::SearchGreatOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchGreatOrEqual(sample)!=-1) printf("元素发现");
   else                                     printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLessOrEqual

在排序数组里搜索小于或等于样本的元素。

```
int  SearchLessOrEqual(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj:: SearchLessOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchLessOrEqual(sample)!=-1) printf("元素发现");
   else                                    printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchFirst

在排序数组里搜索首个与样本相等的元素。

```
int  SearchFirst(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj::SearchFirst(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchFirst(sample)!=-1) printf("元素发现");
   else                              printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# SearchLast

在排序数组里搜索最后一个与模型相等的元素。

```
int  SearchLast(
   CObject*  element      // 样本
   ) const
```

参数

*element*

[输入] 在数组中搜索的样本元素。

返回值

如果成功，发现元素的位置，-1 - 如果项目未发现。

例如：

```
//--- 例程 CArrayObj:: SearchLast(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 排序数组
   array.Sort();
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete array;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(array.SearchLast(sample)!=-1) printf("元素发现");
   else                             printf("元素未发现");
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存数组数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CArrayObj::Save(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加数组元素
   //--- . . .
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载数组数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的二进制文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

注释

当从文件里读取数组并创建每个元素时调用的方法称为 CArrayObj::CreateElement(int)。

例如：

```
//--- 例程 CArrayObj::Load(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用数组元素
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- . . .
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取数组类型标识符。

```
virtual int  Type() const
```

返回值

数组的ID类型（用于CArrayObj-7778）。

例如：

```
//--- 例程 CArrayObj::Type()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取数组类型
   int type=array.Type();
   //--- 删除数组
   delete array;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CList

CList类是存储CObject及其继承类的实例指针的变量动态列表。

## 描述

类CList可供操作 CObject 及其继承者的实例列表。类中实现了在数组里添加 / 插入 / 删除元素的能力，数组排序，在排序的数组里搜索。. 此外，还实现了文件操作方法。

还有某些操作类 CList 的细微之处。这个类具有控制动态内存机制，所以操作列表中的元素时要小心。

内存管理机制的 细节 类似于 CArrayObj中的描述。

## 声明

```
class CList : public CObject
```

## 标称库文件

```
#include <Arrays\List.mqh>
```

## 继承体系

CObject
  CList

## 类方法

| 属性 | |
|---|---|
| FreeMode | 获取删除列表项时的内存管理标志。 |
| FreeMode | 设置删除列表项时的内存管理标志。 |
| Total | 获取列表中元素的数量 |
| IsSorted | 获取列表排序标志 |
| SortMode | 获取排序的版本 |
| 创建方法 | |
| CreateElement | 在列表中创建新元素 |
| 添加方法 | |
| 添加 | 在列表末尾添加元素 |
| Insert | 在列表中按指定位置插入元素 |
| 删除方法 | |
| DetachCurrent | 删除当前位置的列表项但无需将其 "物理" 删除 |
| DeleteCurrent | 删除列表当前位置的元素 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| 删除 | 删除列表指定位置的元素 |
| Clear | 删除全部列表项 |
| 遍历 | |
| IndexOf | 获取列表项的索引 |
| GetNodeAtIndex | 获取指定索引的列表项 |
| GetFirstNode | 获取列表中的首个元素 |
| GetPrevNode | 获取列表中的前一个元素 |
| GetCurrentNode | 获取当前列表项 |
| GetNextNode | 获取下一个列表项 |
| GetLastNode | 获取最后一个列表项 |
| 排序方法 | |
| Sort | 列表排序 |
| MoveToIndex | 移动当前列表项至指定位置 |
| Exchange | 改变列表中元素的座次 |
| 比较方法 | |
| CompareList | 比较列表与另一个列表 |
| 搜索方法 | |
| 搜索 | 在排序列表里搜索与模型相等的元素 |
| 输入/输出 | |
| virtual Save | 保存列表数据至文件 |
| virtual Load | 从文件里加载列表数据 |
| virtual Type | 获取列表类型标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

© 2000-2025, MetaQuotes Ltd.

# FreeMode

获取删除列表项时的内存管理标志。

```
bool  FreeMode() const
```

返回值

内存管理标志。

例如：

```
//--- 例程 CList::FreeMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取释放模式标志
   bool list_free_mode=list.FreeMode();
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# FreeMode

获取删除列表项时的内存管理标志。

```
void  FreeMode(
   bool  mode      // 新值
   )
```

参数

*mode*

[输入] 新的内存管理标志值。

注释

设置内存管理标志 - 这是使用类CList的重点。由于列表元素是动态对象指针，当您从列表删除时判断做什么就很重要。如果标志已设置，当您删除一个列表项时，列表项自动由操作符删除。如果标志未设置，则假设删除对象的指针在用户程序的某处依然使用，且程序随后会将其释放。

如果用户重置了内存管理标志，用户必须了解他们有责任在程序完成前删除列表，因为不这样做的话，在创建新的操作符时原有元素所占用的内存不会被释放。当数据量很大时，它会产生泄露，最终甚至会破坏您的终端。

如果用户没有重置内存管理标志，还有另一个 "暗礁"。使用存储在本地变量某处的列表项指针，当删除列表后将会导致严重错误并令用户程序崩溃。省缺时，内存管理标志被设置，即列表类负责释放内存元素。

例如：

```
//--- 例程 CList::FreeMode(bool)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取释放模式标志
   list.FreeMode(false);
   //--- 使用列表
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Total

获取列表中元素的数量。

```
int  Total() const
```

返回值

列表中元素的数量。

例如：

```
//--- 例程 CList::Total()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查总数
   int total=list.Total();
   //--- 使用列表
   //--- ...
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# IsSorted

获取排序列表标志。

```
bool  IsSorted(
   int  mode=0      // 排序模式
   ) const
```

参数

*mode=0*

[输入] 测试排序版本。

返回值

列表排序标志。如果列表已按指定版本排序?true, 否则?false.

注释

列表已排序征状不可直接修改。标志可通过 Sort (int) 设置，以及由任何添加 / 插入方法重置。

例如：

```
//--- 例程 CList::IsSorted()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查排序
   if(list.IsSorted(0))
     {
      //--- 使用已排序数组的方法
      //--- ...
     }
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

## SortMode

获取排序的版本。

```
int  SortMode() const
```

返回值

排序选项，或者 -1 如果列表未排序。

例如：

```
//--- 例程 CList::SortMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 检查排序模式
   int sort_mode=list.SortMode();
   //--- 使用列表
   //--- ...
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CreateElement

在列表中创建新项目。

```
CObject*  CreateElement()
```

返回值

成功情况下是新创建元素的指针，NULL－如果您未能创建元素。

注释

在 CList 类中的方法 CreateElement() 永远返回 NULL 且不执行任何动作。如果必要，在派生类中，方法 CreateElement() 应予实现。

例如：

```
//--- 例程 CList::CreateElement(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int     size=100;
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- fill list
   for(int i=0;i<size;i++)
     {
      CObject *object=list.CreateElement();
      if(object==NULL)
        {
         printf("元素创建错误");
         delete list;
         return;
        }
      list.Add(object);
     }
   //--- 使用列表
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 添加

在列表末尾添加元素。

```
int  Add(
   CObject*  element      // 添加的元素
   )
```

参数

*element*

[输入] 添加到列表的元素值。

返回值

如果成功，返回添加元素的索引，或在出错时 -1。

注释

如果传递无效指针值 (比如 NULL)，则元素不会被加入列表。

例如：

```
//--- 例程 CList::Add(Cobject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加 100 个元素
   for(int i=0;i<100;i++)
     {
      if(list.Add(new CObject)==-1)
        {
         printf("元素添加错误");
         delete list;
         return;
        }
     }
   //--- 使用列表
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Insert

在列表中按指定位置插入元素。

```
int  Insert(
   CObject*  element,      // 插入的元素
   int       pos           // 位置
   )
```

参数

*element*

[输入] 插入到列表中的元素值。

*pos*

[输入] 插入到列表的位置。

返回值

如果成功，返回插入元素的索引，或在出错时 -1。

注释

如果传递无效指针值 (比如 NULL)，则元素不会被加入列表。

例如：

```
//--- 例程 CList::Insert(CObject*,int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 插入 100 个元素
   for(int i=0;i<100;i++)
     {
      if(list.Insert(new CObject,0)==-1)
        {
         printf("元素插入错误");
         delete list;
         return;
        }
     }
   //--- 使用列表
   //--- . . .
   //--- 删除列表
```

© 2000-2025, MetaQuotes Ltd.

```
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DetachCurrent

抽取列表当前位置的元素但无需将其"物理"删除。

```
CObject*  DetachCurrent()
```

返回值

成功情况下是删除元素的指针，NULL－如果您未能删除元素。

注释

无论内存管理标志处于什么状态，从列表删除时元素并未删除。从使用后释放的成分里抽取列表元素的指针。

例如：

```
//--- 例程 CList::DetachCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.DetachCurrent();
   if(object==NULL)
     {
      printf("分离错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 删除元素
   delete object;
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteCurrent

删除列表当前位置的元素。

```
bool  DeleteCurrent()
```

返回值

true 如果成功，false - 如果您未能删除元素。

注释

如果启用了内存管理，内存占位符释放。

例如：

```
//--- 例程 CList::DeleteCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   if(!list.DeleteCurrent())
     {
      printf("删除错误");
      delete list;
      return;
     }
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

删除列表指定位置的元素。

```
bool  Delete(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 在列表中删除元素的位置。

返回值

true 如果成功，false - 如果您未能删除元素。

注释

如果启用了内存管理，内存占位符释放。

例如：

```
//--- 例程 CList::Delete(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   if(!list.Delete(0))
     {
      printf("删除错误");
      delete list;
      return;
     }
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Clear

删除列表中的所有元素。

```
void  Clear()
```

注释

如果启用内存管理机制，删除的元素被释放。

例如：

```
//--- 例程 CList::Clear()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   //--- 清除列表
   list.Clear();
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# IndexOf

获取列表项的索引。

```
int  IndexOf(
   CObject*  element      // 元素指针
   )
```

参数

*element*

[输入] 列表项的指针。

返回值

列表项的指针，或是 -1。

例如：

```
//--- 例程 CList::IndexOf(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   CObject *object=new CObject;
   if(object==NULL)
     {
      printf("元素创建错误");
      delete list;
      return;
     }
   if(list.Add(object))
     {
      int pos=list.IndexOf(object);
     }
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetNodeAtIndex

获取指定索引的列表项。

```
CObject*  GetNodeAtIndex(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 列表项的索引。

返回值

成功时是列表项的指针，NULL - 如果您未能获取指针。

例如：

```
//--- 例程 CList::GetNodeAtIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetNodeAtIndex(10);
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetFirstNode

获取列表中的首个元素。

```
CObject*  GetFirstNode()
```

返回值

成功时是首个列表项的指针，NULL－如果您未能获取指针。

例如：

```
//--- 例程 CList::GetFirstNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetFirstNode();
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetPrevNode

获取列表中的前一个元素。

```
CObject*  GetPrevNode()
```

返回值

成功时是前一个列表项的指针，NULL - 如果您未能获取指针。

例如：

```
//--- 例程 CList::GetPrevNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetPrevNode();
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetCurrentNode

获取当前列表项。

```
CObject*  GetCurrentNode()
```

返回值

成功时是当前列表项的指针，NULL－如果您未能获取指针。

例如：

```
//--- 例程 CList::GetCurrentNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetCurrentNode();
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetNextNode

获取下一个列表项。

```
CObject*  GetNextNode()
```

返回值

成功时是下一个列表项的指针，NULL - 如果您未能获取指针。

例如：

```
//--- 例程 CList::GetNextNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetNextNode();
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetLastNode

获取列表中的最后一个元素。

```
CObject*  GetLastNode()
```

返回值

成功时是最后一个列表项的指针，NULL - 如果您未能获取指针。

例如：

```
//--- 例程 CList::GetLastNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   CObject *object=list.GetLastNode();
   if(object==NULL)
     {
      printf("获取节点错误");
      delete list;
      return;
     }
   //--- 使用元素
   //--- . . .
   //--- 不能删除元素
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Sort

列表排序。

```
void  Sort(
   int  mode      // 排序模式
   )
```

参数

*mode*

[输入]排序模式。

返回值

无。

注释

列表排序永远是升序。

例如：

```
//--- 例程 CList::Sort(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 排序模式 0
   list.Sort(0);
   //--- 使用列表
   //--- ...
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# MoveToIndex

移动当前列表项至指定位置。

```
bool  MoveToIndex(
   int  pos      // 位置
   )
```

参数

*pos*

[输入] 在列表中移动目标位置。

返回值

true 如果成功，false - 如果您未能移动元素。

例如：

```
//--- 例程 CList::MoveToIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 移动当前节点至开头
   list.MoveToIndex(0);
   //--- 使用列表
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Exchange

改变列表中元素的座次。

```
bool  Exchange(
   CObject*  node1,     // 列表项
   CObject*  node2      // 列表项
   )
```

参数

*node1*

[输入] 列表项

*node2*

[输入] 列表项

返回值

true - 如果成功，false - 如果您未能改变元素排位。

例如：

```
//--- 例程 CList::Exchange(CObject*,CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 交换
   list.Exchange(list.GetFirstNode(),list.GetLastNode());
   //--- 使用列表
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CompareList

比较列表与另一个列表。

```
bool  CompareList(
   CList*  list      // 我们要比较的那个
   )
```

参数

*list*

[输入] CList类实例的指针-用于比较的源元素。

返回值

true 如果列表相等，false - 如果不等。

例如：

```
//--- 例程 CList::CompareList(const CList*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 创建源列表
   CList *src=new CList;
   if(src==NULL)
     {
      printf("对象创建错误");
      delete list;
      return;
     }
   //--- 填充源列表
   //--- . . .
   //--- 与另一列表比较
   bool result=list.CompareList(src);
   //--- 删除列表
   delete src;
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# 搜索

在排序列表里搜索与模型相等的元素。

```
CObject*  Search(
   CObject*  element      // 样本
   )
```

参数

*element*

[输入] 在列表中搜索的样本单元。

返回值

如果成功，发现列表项的指针，-1 - 如果列表项未发现。

例如：

```
//--- 例程 CList::Search(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   //--- 列表排序
   list.Sort(0);
   //--- 创建样本
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("样本创建错误");
      delete list;
      return;
     }
   //--- 设置样本属性
   //--- . . .
   //--- 搜索元素
   if(list.Search(sample)!=NULL) printf("元素发现");
   else                          printf("元素未发现");
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

## Save

保存列表数据至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CList::Save(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 添加列表元素
   //--- . . .
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Save(file_handle))
        {
         //--- file save error
         printf("文件保存: 错误 %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 删除列表
   delete list;
```

© 2000-2025, MetaQuotes Ltd.

```
}
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载列表数据。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

注释

当从文件里读取列表并创建每个元素时调用的方法称为 CList::CreateElement()。

例如：

```
//--- 例程 CLoad::Load(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Load(file_handle))
        {
         //--- 文件加载出错
         printf("文件加载: 错误 %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用列表元素
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- . . .
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取列表类型标识符。

```
virtual int  Type()
```

返回值

类型标识符 (对应 CList-7779)。

例如：

```
//--- 例程 CList::Type()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("对象创建错误");
      return;
     }
   //--- 获取列表类型
   int type=list.Type();
   //--- 删除列表
   delete list;
  }
```

© 2000-2025, MetaQuotes Ltd.

# CTreeNode

类CTreeNode是CTree二叉树的节点类。

## 描述

类CTreeNode可供操作CTree二叉树的节点。在类中，实现了遍历二叉树选项。此外，还实现了文件操作方法。

## 声明

```
class CTreeNode : public CObject
```

## 标称库文件

```
#include <Arrays\TreeNode.mqh>
```

## 继承体系

CObject

CTreeNode

直接分支

CTree

## 类方法

| 属性 | |
|---|---|
| Owner | 获取/设置节点所有者的指针 |
| Left | 获取/设置左节点的指针 |
| Right | 获取/设置右节点的指针 |
| Balance | 获取节点平衡 |
| BalanceL | 获取节点左子树平衡 |
| BalanceR | 获取节点右子树平衡 |
| 创建一个新元素 | |
| CreateSample | 创建一个新节点实例 |
| 比较 | |
| RefreshBalance | 重新计算节点平衡 |
| 搜索 | |
| GetNext | 获取下一个节点的指针 |
| 输入/输出 | |
| SaveNode | 保存节点数据至文件 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| LoadNode | 从文件里加载节点数据 |
| virtual Type | 获取节点类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Compare

CTreeNode类的子类树得到实际应用。

类CTreeNode的子类必须预定义方法：CreateSample用来创建类CTreeNode的子类新实例，Compare比较类CTreeNode子类的关键字段值，Type（如果有必要标识一个节点），SaveNode和LoadNode（如果有必要操作文件）。

我们来研究一个CTree子类的例子。

```
//+------------------------------------------------------------------+
//|                                                   MyTreeNode.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                        https://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\TreeNode.mqh>
//+------------------------------------------------------------------+
//| 描述 自 CTreeNode 派生的类                                       |
//+------------------------------------------------------------------+
//| 类 CMyTreeNode。                                                 |
//| 目的: 二叉树元素类                                               |
//|              类 CTreeNode 的子类                                 |
//+------------------------------------------------------------------+
class CMyTreeNode : public CTreeNode
  {
protected:
   //--- 用户数据
   long              m_long;           // 长整数型关键字段
   double            m_double;         // 双精度型自定义变量
   string            m_string;         // 字符串型自定义变量
   datetime          m_datetime;       // 日期时间型自定义变量

public:
                     CMyTreeNode();
   //--- 访问用户数据的方法
   long              GetLong(void)                { return(m_long); }
   void              SetLong(long value)          { m_long=value;  }
   double            GetDouble(void)              { return(m_double); }
   void              SetDouble(double value)      { m_double=value;  }
   string            GetString(void)              { return(m_string); }
```

© 2000-2025, MetaQuotes Ltd.

```
   void              SetString(string value)       { m_string=value;   }
   datetime          GetDateTime(void)             { return(m_datetime); }
   void              SetDateTime(datetime value)   { m_datetime=value;   }
   //--- 操作文件的方法
   virtual bool      Save(int file_handle);
   virtual bool      Load(int file_handle);
protected:
   virtual int       Compare(const CObject *node,int mode);
   //--- 创建类实例的方法
   virtual CTreeNode* CreateSample();
  };
//+------------------------------------------------------------------+
//| CMyTreeNode 类构造器。                                           |
//| 输入:  无。                                                      |
//| 输出: 无。                                                       |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
void CMyTreeNode::CMyTreeNode()
  {
//--- 初始化用户数据
   m_long         =0;
   m_double       =0.0;
   m_string       ="";
   m_datetime     =0;
  }
//+------------------------------------------------------------------+
//| 按照指定算法比较其它三个节点。                                   |
//| 输入:  节点 - 比较的元素数组,                                    |
//|          mode - 比较算法标识符。                                 |
//| 输出: 比较结果 (>0,0,<0)。                                       |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
int CMyTreeNode::Compare(const CObject *node,int mode)
  {
//--- 参数 mode 被忽略, 因为数构造算法只有一个
   int res=0;
//--- 显式类型转换
   CMyTreeNode *n=node;
   res=(int)(m_long-n.m_long);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| 创建新的类实例。                                                 |
//| 输入:  无。                                                      |
//| 输出: 类 CMyTreeNode 的新实例指针。                              |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
CTreeNode* CMyTreeNode::CreateSample()
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   CMyTreeNode *result=new CMyTreeNode;
//---
   return(result);
  }
//+------------------------------------------------------------------+
//| 写树节点数据至文件。                                             |
//| 输入:  file_handle -以写模式打开的二进制文件句柄。               |
//| 输出: 如果成功 true, 否则 false。                                |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CMyTreeNode::Save(int file_handle)
  {
   uint i=0,len;
//--- 检查
   if(file_handle<0) return(false);
//--- 写用户数据
//--- 写长整数型自定义变量
   if(FileWriteLong(file_handle,m_long)!=sizeof(long))          return(false);
//--- writing custom variable of type double
   if(FileWriteDouble(file_handle,m_double)!=sizeof(double))    return(false);
//--- 写字符串型自定义变量
   len=StringLen(m_string);
//--- 写字符串长度
   if(FileWriteInteger(file_handle,len,INT_VALUE)!=INT_VALUE)   return(false);
//--- 写字符串
   if(len!=0 && FileWriteString(file_handle,m_string,len)!=len) return(false);
//--- 写日期时间型自定义变量
   if(FileWriteLong(file_handle,m_datetime)!=sizeof(long))      return(false);
//---
   return(true);
  }
//+------------------------------------------------------------------+
//| 从文件里读书节点数据。                                           |
//| 输入:  file_handle -以读模式打开的二进制文件句柄。               |
//| 输出: 如果成功 true, 否则 false。                                |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CMyTreeNode::Load(int file_handle)
  {
   uint i=0,len;
//--- 检查
   if(file_handle<0) return(false);
//--- reading
   if(FileIsEnding(file_handle)) return(false);
//--- 读字符型自定义变量
//--- 读长整数型自定义变量
   m_long=FileReadLong(file_handle);
//--- 读双精度型自定义变量
```

© 2000-2025, MetaQuotes Ltd.

```
   m_double=FileReadDouble(file_handle);
//--- 读字符串型自定义变量
//--- 读字符串长度
   len=FileReadInteger(file_handle,INT_VALUE);
//--- 读字符串
   if(len!=0) m_string=FileReadString(file_handle,len);
   else       m_string="";
//--- 读日期时间型自定义变量
   m_datetime=FileReadLong(file_handle);
//---
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

## Owner

获取节点所有者的指针。

```
CTreeNode*  Owner()
```

返回值

节点所有者的指针。

## Owner

设置节点所有者的指针。

```
void  Owner(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[属如] 节点所有者指针的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Left

获取左节点的指针。

```
CTreeNode*  Left()
```

返回值

左节点的指针。

## Left

设置左节点的指针。

```
void  Left(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 左节点指针的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Right

获取右节点的指针。

```
CTreeNode*  Right()
```

返回值

右节点的指针。

## Right

设置右节点的指针。

```
void  Right(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 右节点的指针新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Balance

获取节点平衡。

```
int  Balance() const
```

返回值

节点平衡。

© 2000-2025, MetaQuotes Ltd.

# BalanceL

获取节点左子树平衡。

```
int  BalanceL() const
```

返回值

节点左子树平衡。

© 2000-2025, MetaQuotes Ltd.

# BalanceR

获取节点右子树平衡。

```
int  BalanceR() const
```

返回值

节点右子树平衡。

© 2000-2025, MetaQuotes Ltd.

# CreateSample

创建一个新节点样本。

```
virtual CTreeNode*  CreateSample()
```

返回值

新节点样本指针或 NULL。

© 2000-2025, MetaQuotes Ltd.

# RefreshBalance

重新计算节点平衡。

```
int  RefreshBalance()
```

返回值

节点平衡。

© 2000-2025, MetaQuotes Ltd.

# GetNext

获取下一个节点的指针。

```
CTreeNode*  GetNext(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 开始搜索的节点。

返回值

下一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

# SaveNode

写节点数据至文件。

```
bool  SaveNode(
   int  file_handle      // 句柄
   )
```

参数

*file_handle*

[输入] 以写模式打开的二进制文件句柄。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# LoadNode

从文件里读取节点数据。

```
bool  LoadNode(
   int          file_handle,     // 句柄
   CTreeNode*   main             // 节点
   )
```

参数

*file_handle*

[输入] 以读模式打开的二进制文件句柄。

*main*

[输入] 数据节点。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Type

获取节点类型的标识符。

```
virtual int  Type() const
```

返回值

节点类型的标识符。

© 2000-2025, MetaQuotes Ltd.

# CTree

CTree 类是 CTreeNode 类及其子类的样本二叉树。

## 描述

类 CTree 可供操作 CTreeNode 类及其子类的样本二叉树。在类中，实现了三元素添加/插入/删除选项以及搜索二叉树。此外，还实现了文件操作方法。

注释，在类 CTree 没有实现内存管理机制（这点与类 CList 和 CArrayObj 不同）。所有树节点被删除时内存一并释放。

## 声明

```
class CTree : public CTreeNode
```

## 标称库文件

```
#include <Arrays\Tree.mqh>
```

## 继承体系

CObject
  CTreeNode
    CTree

## 类方法

| 属性 | |
|---|---|
| Root | 获取树的根节点 |
| 创建一个新元素 | |
| CreateElement | 创建一个新节点实例 |
| 填充 | |
| Insert | 在树中添加节点 |
| 删除 | |
| Detach | 从树里分离一个指定节点 |
| 删除 | 从树里删除一个指定节点 |
| Clear | 从树里删除所有节点 |
| 搜索 | |
| Find | 在树里按样本搜索一个节点 |
| 输入/输出 | |
| virtual Save | 保存所有树数据至文件 |
| virtual Load | 从文件里加载树数据 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| virtual Type | 获取树类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CTreeNode

Parent, Parent, Left, Left, Right, Right, Balance, BalanceL, BalanceR, RefreshBalance, GetNext, SaveNode, LoadNode

CTreeNode类的子类树 -子类CTree得到实际应用。

类CTree的子类必须有一个预定义方法 CreateElement 用于创建子类CTreeNode的新样本。

我们来研究一个 CTree 子类的例子。

```
//+------------------------------------------------------------------+
//|                                                      MyTree.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//| www.metaquotes.net |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\Tree.mqh>
#include "MyTreeNode.mqh"
//---
input int extCountedNodes = 100;
//+------------------------------------------------------------------+
//| 描述类 CMyTree derived from CTree。                          |
//+------------------------------------------------------------------+
//| Class CMyTree.                                                   |
//| 目的: 构造并遍历二叉树。    |
//+------------------------------------------------------------------+
class CMyTree : public CTree
  {
public:
   //--- 以自定义数据搜索二叉树的方法
   CMyTreeNode*         FindByLong(long find_long);
   //--- 创建树元素的方法
   virtual CTreeNode *CreateElement();
  };
//---
CMyTree MyTree;
//+------------------------------------------------------------------+
//| 创建新的树节点。                                          |
//| 输入:  无。                                                  |
//| 输出: 成功则为新树节点指针, 或是 NULL。                |
```

```
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
CTreeNode *CMyTree::CreateElement()
  {
   CMyTreeNode *node=new CMyTreeNode;
//---
   return(node);
  }
//+------------------------------------------------------------------+
//| 按照值 m_long 在列表里搜索元素。                                 |
//| 输入:  find_long - 搜索值。                                      |
//| 输出: 发现的元素指针, 或是 NULL。                                |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
CMyTreeNode* CMyTree::FindByLong(long find_long)
  {
   CMyTreeNode *res=NULL;
   CMyTreeNode *node;
//--- 创建树节点, 传递搜索参数
   node=new CMyTreeNode;
   if(node==NULL) return(NULL);
   node.SetLong(find_long);
//---
   res=Find(node);
   delete node;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| 脚本 "类 CMyTree 测试"                                           |
//+------------------------------------------------------------------+
//---  用于字符串初始化的数组
string str_array[11]={"p","oo","iii","uuuu","yyyyy","ttttt","rrrr","eee","ww","q","999
//---
int OnStart() export
  {
   int          i;
   uint         pos;
   int          beg_time,end_time;
   CMyTreeNode *node; //--- 类 CMyTreeNode 样本的临时指针
//---
   printf("开始测试 %s。",__FILE__);
//--- 用数量为 extCountedNodes 的 MyTreeNode 类样本填满 MyTree。
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i++)
     {
      node=MyTree.CreateElement();
      if(node==NULL)
        {
```

© 2000-2025, MetaQuotes Ltd.

```
         //--- 紧急出口
         printf("%s (%4d): 创建错误",__FILE__,__LINE__);
         return(__LINE__);
        }
      NodeSetData(node,i);
      node.SetLong(i);
      MyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("填充 MyTree 时间是 %d 毫秒。",end_time-beg_time);
//--- 创建临时树 TmpMyTree。
   CMyTree TmpMyTree;
//--- 分离 50% 树元素 (所有偶数)
//--- 并将它们添加到临时树 TmpMyTree。
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i+=2)
     {
      node=MyTree.FindByLong(i);
      if(node!=NULL)
         if(MyTree.Detach(node)) TmpMyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("从 MyTree 删除 %d 元素的时间是 %d 毫秒。",extCountedNodes/2,end_time-beg_time);
//--- 返回分离
   node=TmpMyTree.Root();
   while(node!=NULL)
     {
      if(TmpMyTree.Detach(node)) MyTree.Insert(node);
      node=TmpMyTree.Root();
     }
//--- 检查操作方法 Save(int file_handle);
   int file_handle;
   file_handle=FileOpen("MyTree.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!MyTree.Save(file_handle))
        {
         //--- 写文件错误
         //--- 紧急出口
         printf("%s: 错误 %d 为 %d!",__FILE__,GetLastError(),__LINE__);
         //--- 离开前关闭文件!!!
         FileClose(file_handle);
         return(__LINE__);
        }
      FileClose(file_handle);
     }
//--- 检查操作方法 Load(int file_handle);
   file_handle=FileOpen("MyTree.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       if(!TmpMyTree.Load(file_handle))
         {
          //--- 读文件错误
          //--- 紧急出口
          printf("%s: 错误 %d 为 %d!",__FILE__,GetLastError(),__LINE__);
          //--- 离开前关闭文件!!!
          FileClose(file_handle);
          return(__LINE__);
         }
       FileClose(file_handle);
      }
//---
   MyTree.Clear();
   TmpMyTree.Clear();
//---
   printf("测试结束 %s。OK!",__FILE__);
//---
   return(0);
  }
//+------------------------------------------------------------------+
//| 节点内容输出至日志功能                                           |
//+------------------------------------------------------------------+
void NodeToLog(CMyTreeNode *node)
  {
   printf("   %I64d,%f,'%s','%s'",
               node.GetLong(),node.GetDouble(),
               node.GetString(),TimeToString(node.GetDateTime()));
  }
//+------------------------------------------------------------------+
//| 用随机值 "填充" 节点功能                                         |
//+------------------------------------------------------------------+
void NodeSetData(CMyTreeNode *node,int mode)
  {
   if(mode%2==0)
     {
      node.SetLong(mode*MathRand());
      node.SetDouble(MathPow(2.02,mode)*MathRand());
     }
   else
     {
      node.SetLong(mode*(long)(-1)*MathRand());
      node.SetDouble(-MathPow(2.02,mode)*MathRand());
     }
   node.SetString(str_array[mode%10]);
   node.SetDateTime(10000*mode);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Root

获取树的根节点。

```
CTreeNode*  Root() const
```

返回值

树的根节点指针。

© 2000-2025, MetaQuotes Ltd.

# CreateElement

创建新的节点实例。

```
virtual CTreeNode*  CreateElement()
```

返回值

节点新实例的指针或 NULL。

© 2000-2025, MetaQuotes Ltd.

# Insert

在树中添加节点。

```
CTreeNode*  Insert(
   CTreeNode*  new_node      // 节点
   )
```

参数

*new_node*

[输入]在树中插入节点的指针。

返回值

节点所有者的指针或 NULL。

© 2000-2025, MetaQuotes Ltd.

# Detach

从树里分离一个指定节点。

```
bool  Detach(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 要分离的节点指针。

返回值

true 如果成功，否则 false。

注释

当节点分离后指针不会释放。树是平衡的。

© 2000-2025, MetaQuotes Ltd.

# 删除

从树里删除一个指定节点。

```
bool  Delete(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 要删除的节点指针。

返回值

true 如果成功，否则 false。

注释

当节点删除后指针被释放。树是平衡的。

© 2000-2025, MetaQuotes Ltd.

# Clear

从树里删除所有节点。

```
void  Clear()
```

返回值

无。

注释

当节点删除后指针被释放。

© 2000-2025, MetaQuotes Ltd.

# Find

在树里按样本搜索一个节点。

```
CTreeNode*  Find(
   CTreeNode*  node      // 节点
   )
```

参数

*node*

[输入] 包含数据的节点-搜索样本。

返回值

发现的节点指针或 NULL。

© 2000-2025, MetaQuotes Ltd.

## Save

写树数据至文件。

```
virtual bool  Save(
   int  file_handle      // 句柄
   )
```

参数

*file_handle*

[输入] 以写模式打开的二进制文件句柄。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载树数据。

```
virtual bool  Load(
   int  file_handle      // 句柄
   )
```

参数

*file_handle*

[输入] 以读模式打开的二进制文件句柄。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Type

获取树类型的标识符。

```
virtual int  Type() const
```

返回值

树类型的标识符。

© 2000-2025, MetaQuotes Ltd.

# 通用数据集

程序库提供了定义通用集的类和接口，允许用户创建增强型类型集。这些集合提供了比非通用类型集更大的便利性和数据处理性能。

程序库可以在程序端工作目录的Include\Generic文件夹中得到。

对象：

| 对象 | 描述 | 类型 |
|---|---|---|
| ICollection | 实施通用数据集的接口 | INTERFACE |
| IEqualityComparable | 用于实施可以比较的对象的接口 | INTERFACE |
| IComparable | 用于实施可以通过“大于，小于或等于”进行比较的对象的接口 | INTERFACE |
| IComparer | 用于实施比较两个T类型对象的通用类的接口，比较其中一个是否“大于，小于或等于”另一个对象 | INTERFACE |
| IEqualityComparer | 用于实施比较两个T类型对象相等性的通用类的接口 | INTERFACE |
| IList | 实施通用数据列表的接口 | INTERFACE |
| IMap | 实施键/值对的通用集的接口 | INTERFACE |
| ISet | 实施通用数据集的接口 | INTERFACE |
| CDefaultComparer | 基于比较全局方法实施IComparer<T>通用接口的辅助类 | CLASS |
| CDefaultEqualityComparer | 使用Equals<T>和GetHashCode全局方法实施IEqualityComparer<T>通用接口的辅助类 | CLASS |
| CArrayList | 实施IList<T>接口的通用类 | CLASS |
| CKeyValuePair | 这个类实施键/值对 | CLASS |
| CHashMap | 实施IMap<TKey, TValue>接口的通用类 | CLASS |
| CHashSet | 实施ISet<T>接口的通用类 | CLASS |
| CLinkedListNode | 实施CLinkedListNode<T>类的辅助类 | CLASS |
| CLinkedList | 实施ICollection<T>接口的通用类 | CLASS |
| CQueue | 实施ICollection<T>接口的通用类 | CLASS |
| CRedBlackTreeNode | 用于实施CRedBlackTree<T>类的辅助类 | CLASS |
| CRedBlackTree | 实施ICollection<T>接口的通用类 | CLASS |

© 2000-2025, MetaQuotes Ltd.

| 对象 | 描述 | 类型 |
| --- | --- | --- |
| CSortedMap | 实施IMap<TKey,TValue>接口的通用类 | CLASS |
| CSortedSet | 实施ISet<T>接口的通用类 | CLASS |
| CStack | 实施ICollection<T>接口的通用类 | CLASS |

全局方法：

| 方法 | 描述 |
| --- | --- |
| ArrayBinarySearch | 使用IComparable<T>接口，搜索升序一维数组指定值来比较元素 |
| ArrayIndexOf | 搜索一维数组中第一次出现的值 |
| ArrayLastIndexOf | 搜索一维数组中最后出现的值 |
| ArrayReverse | 改变一维数组中的元素序列 |
| Compare | 比较两个值的大小，其中一个是否大于，小于或等于另一个值 |
| Equals | 比较两个值的相等性 |
| GetHashCode | 计算散列码值 |

© 2000-2025, MetaQuotes Ltd.

# ICollection<T>

ICollection<T>是实施通用数据集的接口。

## 描述

ICollection<T>接口确定处理集合的基本方法，包括元素计数方法，清空集合方法，添加或删除元素方法等。

## 声明

```
template<typename T>
interface ICollection
```

## 标题

```
#include <Generic\Interfaces\ICollection.mqh>
```

## 继承体系

ICollection

直接分支

CLinkedList, CQueue, CRedBlackTree, CStack, IList, IMap, ISet

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到集合 |
| Count | 返回集合中的元素数量 |
| Contains | 确定集合是否包含指定值的元素 |
| CopyTo | 从指定索引开始复制集合的所有元素到指定数组 |
| Clear | 移除集合的所有元素 |
| Remove | 移除集合中第一次出现的指定元素 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到集合。

```
bool Add(
   T  value     // 元素的值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回集合中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定集合是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果集合中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制集合的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用于编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写集合元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除集合的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除集合中第一次出现的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IEqualityComparable<T>

IEqualityComparable<T>是实施可以比较的对象的接口。

## 描述

IEqualityComparable<T>接口定义了检索当前对象散列码和检查是否等于同类型的另一个对象的方法。

## 声明

```
template<typename T>
interface IEqualityComparable
```

## 标题

```
#include <Generic\Interfaces\IEqualityComparable.mqh>
```

## 继承体系

IEqualityComparable

直接分支

IComparable

## 类方法

| 方法 | 描述 |
| --- | --- |
| Equals | 比较当前对象和指定值 |
| HashCode | 计算当前对象的散列码值 |

© 2000-2025, MetaQuotes Ltd.

# Equals

比较当前对象和指定值。

```
bool Equals(
   T  value     // 要比较的值
   );
```

参数

*value*

[in] 要比较当前对象的值。

返回值

如果对象相等，返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# HashCode

计算当前对象的散列码值。

```
int HashCode();
```

返回值

返回散列代码。

© 2000-2025, MetaQuotes Ltd.

# IComparable<T>

IComparable<T>是实施两个对象的接口，用于比较其中一个是否大于，小于或等于另一个对象

## 描述

IComparable<T>接口定义了比较当前对象与另一个同类型对象的方法，在此基础上可以排序这些对象的集合。

## 声明

```
template<typename T>
interface IComparable : public IEqualityComparable<T>
```

## 标题

```
#include <Generic\Interfaces\IComparable.mqh>
```

## 继承体系

IEqualityComparable

IComparable

直接分支

CKeyValuePair

## 类方法

| 方法 | 描述 |
| --- | --- |
| Compare | 比较当前对象和指定值 |

© 2000-2025, MetaQuotes Ltd.

# Compare

比较当前对象和指定值。

```
int Compare(
   T  value     // 要比较的值
   );
```

参数

*value*

[in] 要比较当前对象的值。

返回值

返回表示当前对象与过去对象比率的数字：

- 如果结果小于0，表示当前对象小于过去对象
- 如果结果等于0，表示当前对象等于过去对象
- 如果结果大于0，表示当前对象大于过去对象

© 2000-2025, MetaQuotes Ltd.

# IComparer<T>

IComparer<T>是实施比较两个T类型对象的通用类的接口，比较其中一个是否大于，小于或等于另一个对象。

## 描述

IComparer<T>接口定义了比较两个T类型对象的方法，在此基础上可以排序这些对象的集合。

## 声明

```
template<typename T>
interface IComparer
```

## 标题

```
#include <Generic\Interfaces\IComparer.mqh>
```

## 继承体系

IComparer

直接分支

CDefaultComparer

## 类方法

| 方法 | 描述 |
| --- | --- |
| Compare | 比较类型T的两个值 |

© 2000-2025, MetaQuotes Ltd.

# Compare

比较类型T的两个值。

```
int Compare(
   T  x,      //第一个值
   T  y       //第二个值
   );
```

参数

*x*

[in] 要比较的第一个值。

*y*

[in] 要比较的第一个值。

返回值

返回表示相比较的两个值比率的数字：

- 如果结果小于0，x则小于y (x<y)
- 如果结果等于0，x则等于y (x=y)
- 如果结果大于0，x则大于y (x>y)

© 2000-2025, MetaQuotes Ltd.

# IEqualityComparer<T>

IEqualityComparer<T>是实施比较两个T类型对象的通用类的接口。

## 描述

IEqualityComparer<T>接口定义了检索T类型对象散列码和检查两个T类型对象是否相等的方法。

## 声明

```
template<typename T>
interface IEqualityComparer
```

## 标题

```
#include <Generic\Interfaces\IEqualityComparer.mqh>
```

## 继承体系

IEqualityComparer

直接分支

CDefaultEqualityComparer

## 类方法

| 方法 | 描述 |
| --- | --- |
| Equals | 比较类型T的两个值 |
| HashCode | 基于T类型对象计算散列代码值 |

© 2000-2025, MetaQuotes Ltd.

# Equals

比较类型T的两个值。

```
bool Equals(
   T  x,      //第一个值
   T  y       //第二个值
   );
```

参数

*x*

[in] 要比较的第一个值。

*y*

[in] 要比较的第二个值。

返回值

如果两个值相等，返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# HashCode

基于T类型对象计算散列代码值。

```
int HashCode(
   T  value     // 用于计算的对象
   );
```

参数

*value*

[in]您想要获得散列代码的对象。

返回值

返回散列代码。

© 2000-2025, MetaQuotes Ltd.

# IList<T>

IList<T>是实施通用数据列表的接口。

## 描述

IList<T>接口定义了处理列表的基本方法，例如根据索引访问元素，搜索和删除元素，排列顺序等。

## 声明

```
template<typename T>
interface IList : public ICollection<T>
```

## 标题

```
#include <Generic\Interfaces\IList.mqh>
```

## 继承体系

ICollection

IList

直接分支

CArrayList

## 类方法

| 方法 | 描述 |
| --- | --- |
| TryGetValue | 获得指定索引的列表元素 |
| TrySetValue | 更改指定索引的列表值 |
| Insert | 将元素插入到指定索引的列表中 |
| IndexOf | 搜索列表中第一次出现的值 |
| LastIndexOf | 搜索列表中最后出现的值 |
| RemoveAt | 移除指定索引的列表元素 |

© 2000-2025, MetaQuotes Ltd.

# TryGetValue

获得指定索引的列表元素。

```
bool TryGetValue(
   const int   index,      // 元素索引
   T&          value       // 用来编写的变量
   );
```

参数

*index*

[in] 列表的元素索引。

*&value*

[out] 将要编写列表元素指定值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrySetValue

更改指定索引的列表值。

```
bool TrySetValue(
   const int   index,      //元素索引
   T           value       // 新值
   );
```

参数

*index*

[in] 列表的元素索引。

*value*

[in] 分配给指定列表元素的新值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Insert

将元素插入到指定索引的列表中。

```
bool Insert(
   const int  index,      // 要插入的索引位置
   T          item        //要被插入的值
   );
```

参数

*index*

[in] 要插入的索引位置。

*item*

[in] 要在指定索引位置插入的值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IndexOf

搜索列表中第一次出现的值。

```
int IndexOf(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

返回找到的第一个元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# LastIndexOf

搜索列表中最后出现的值。

```
int LastIndexOf(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

返回最后找到的元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# RemoveAt

移除指定索引的列表元素。

```
bool RemoveAt(
   const int  index     // 元素索引
   );
```

参数

*index*

[in] 您想删除的元素索引。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IMap<TKey, TValue>

IMap<TKey, TValue>是实施键/值对的通用集的接口。

## 描述

IMap<TKey, TValue>接口定义了处理数据存储为键/值对的集合的基本方法。

## 声明

```
template<typename TKey, typename TValue>
interface IMap : public ICollection<TKey>
```

## 标题

```
#include <Generic\Interfaces\IMap.mqh>
```

## 继承体系

ICollection

IMap

直接分支

CHashMap, CSortedMap

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加键/值对到集合 |
| Contains | 确定集合是否包含指定键的键/值表 |
| Remove | 移除集合中第一次出现的键/值对 |
| TryGetValue | 获得集合中指定键的元素 |
| TrySetValue | 在指定键，更改集合中键/值对的值 |
| CopyTo | 从指定索引开始，复制来自集合的所有键/值对到指定数组 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加键/值对到集合。

```
bool Add(
   TKey    key,       //键
   TValue  value      // 值
   );
```

参数

*key*

[in] 键。

*value*

[in] 值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定集合是否包含指定键的键/值表。

```
bool Contains(
   TKey    key,       //键
   TValue  value      // 值
   );
```

参数

*key*

[in] 键。

*value*

[in] 值。

返回值

如果集合包含指定键和值的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Remove

移除集合中第一次出现的键/值对。

```
bool Remove(
   TKey  key      // 键
   );
```

参数

*key*

[in] 键。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TryGetValue

获得集合中指定键的元素。

```
bool TryGetValue(
   TKey      key,       //键
   TValue&   value      //用来编写值的变量
   );
```

参数

*key*

[in] 键。

*&value*

[out] 将要编写键/值对指定值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrySetValue

在指定键，更改集合中键/值对的值。

```
bool TrySetValue(
   TKey    key,       // 键
   TValue  value      // 新值
   );
```

参数

*key*

[in] 键。

*value*

[in] 分配给指定键-值对的新值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始，复制来自集合的所有键/值对到指定数组。

```
int CopyTo(
   TKey&       dst_keys[],        //用于编写键的数组
   TValue&     dst_values[],      //用于编写值的数组
   const int   dst_start=0        //用于编写的索引起始点
   );
```

参数

*&dst_keys[]*

[out] 将要编写集合所有键的数组。

*&dst_values[]*

[out] 将要编写集合相关键的值的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制键/值对的数量。

© 2000-2025, MetaQuotes Ltd.

# ISet<T>

ISet<T>是实施通用数据集的接口。

## 描述

ISet接口定义了处理集合的基本方法，例如：集合的联合和交集，严格和非严格子集的定义等。

## 声明

```
template<typename T>
interface ISet : public ICollection<T>
```

## 标题

```
#include <Generic\Interfaces\ISet.mqh>
```

## 继承体系

ICollection
  ISet

直接分支

CHashSet, CSortedSet

## 类方法

| 方法 | 描述 |
| --- | --- |
| ExceptWith | 产生当前集合和过去集合之间的操作差异（数组） |
| IntersectWith | 产生当前集合和过去集合的操作交集（数组） |
| SymmetricExceptWith | 产生当前集合和过去集合之间的操作对称差（数组） |
| UnionWith | 产生当前集合和过去集合的结合（数组） |
| IsProperSubsetOf | 确定当前集合是否为指定集合或数组的适当子集 |
| IsProperSupersetOf | 确定当前集合是否为指定集合或数组的适当超集合 |
| IsSubsetOf | 确定当前集合是否为指定集合或数组的子集 |
| IsSupersetOf | 确定当前集合是否为指定集合或数组的超集合 |
| Overlaps | 确定当前集合是否与指定集合或数组重复 |
| SetEquals | 确定当前集合是否包含指定集合或数组的所有元素 |

© 2000-2025, MetaQuotes Ltd.

# ExceptWith

产生当前集合和过去集合之间的操作差异 (数组)。从当前集合 (数组)移除指定集合 (数组)中存在的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
void ExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void ExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 当前集合以外的集合。

*&collection[]*

[in] 当前集合以外的数组。

注意

这个结果被写入当前集合 (数组)。

© 2000-2025, MetaQuotes Ltd.

# IntersectWith

产生当前集合和过去集合的操作交集 (数组)。更改当前集合到只包含在指定集合中存在的元素 (数组)。

这个版本处理了实施 ICollection<T>接口的集合。

```
void IntersectWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void IntersectWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合相交的集合。

*&collection[]*

[in] 与当前集合相交的数组。

注意

这个结果被写入当前集合 (数组)。

© 2000-2025, MetaQuotes Ltd.

# SymmetricExceptWith

产生当前集合和过去集合之间的操作对称差（数组）。更改当前集合到只包含在源对象或指定集合中存在的元素（数组），但不是同时存在两者中。

这个版本处理了实施 ICollection<T>接口的集合。

```
void SymmetricExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void SymmetricExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 产生对称差的集合。

*&collection[]*

[in] 产生对称差的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# UnionWith

产生当前集合和过去集合的结合（数组）。将从指定集合（数组）缺失的元素添加到当前集合（数组）。

这个版本处理了实施 ICollection<T>接口的集合。

```
void UnionWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void UnionWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合一致的集合。

*&collection[]*

[in] 与当前集合一致的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# IsProperSubsetOf

确定当前集合是否为指定集合或数组的适当子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为适当子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsProperSupersetOf

确定当前集合是否为指定集合或数组的适当超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为适当超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSubsetOf

确定当前集合是否为指定集合或数组的子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSupersetOf

确定当前集合是否为指定集合或数组的超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Overlaps

确定当前集合是否与指定集合或数组重复。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool Overlaps(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool Overlaps(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 确定重复的集合。

*&collection[]*

[in] 确定重复的数组。

返回值

如果当前集和集合或数组重复，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# SetEquals

确定当前集合是否包含指定集合或数组的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool SetEquals(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool SetEquals(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 比较元素的集合。

*&collection[]*

[in] 比较元素的集合。

返回值

如果当前集包括指定集合或数组的所有元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CDefaultComparer<T>

CDefaultComparer<T>是基于比较全局方法实施 IComparer<T>通用接口的辅助类。

## 描述

CDefaultComparer<T>类默认用在通用数据集，除非用户默默使用其他实施 IComparer<T>接口的类。

## 声明

```
template<typename T>
class CDefaultComparer : public IComparer<T>
```

## 标题

```
#include <Generic\Internal\DefaultComparer.mqh>
```

## 继承体系

IComparer

CDefaultComparer

## 类方法

| 方法 | 描述 |
| --- | --- |
| Compare | 比较类型T的两个值 |

© 2000-2025, MetaQuotes Ltd.

# Compare

比较类型T的两个值。

```
int Compare(
   T  x,      //第一个值
   T  y       //第二个值
   );
```

参数

*x*

[in] 要比较的第一个值。

*y*

[in] 要比较的第二个值。

返回值

返回表示相比较的两个值比率的数字：

- 如果结果小于0，x则小于y (x<y)
- 如果结果等于0，x则等于y (x=y)
- 如果结果大于0，x则大于y (x>y)

注意

x和y两个值基于T类型的其中一种重载比较全局方法进行比较。

© 2000-2025, MetaQuotes Ltd.

# CDefaultEqualityComparer<T>

CDefaultEqualityComparer<T>是基于Equals<T>和GetHashCode全局方法实施IEqualityComparer<T>通用接口的辅助类。

## 描述

CDefaultEqualityComparer<T> 类默认用在通用数据集，除非用户默默使用其他实施IEqualityComparer<T>接口的类。

## 声明

```
template<typename T>
class CDefaultEqualityComparer : public IEqualityComparer<T>
```

## 标题

```
#include <Generic\Internal\DefaultEqualityComparer.mqh>
```

## 继承体系

IEqualityComparer

CDefaultEqualityComparer

## 类方法

| 方法 | 描述 |
| --- | --- |
| Equals | 比较类型T的两个值 |
| HashCode | 基于T类型对象计算散列代码值 |

© 2000-2025, MetaQuotes Ltd.

# Equals

比较类型T的两个值。

```
bool Equals(
   T  x,      // 第一个值
   T  y       //第二个值
   );
```

参数

*x*

[in] 要比较的第一个值。

*y*

[in] 要比较的第二个值。

返回值

如果两个值相等，返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# HashCode

基于T类型对象计算散列代码值。

```
int HashCode(
   T  value     // 用于计算的对象
   );
```

参数

*value*

[in]您想要获得散列代码的对象。

返回值

返回散列代码。

© 2000-2025, MetaQuotes Ltd.

# CRedBlackTreeNode<T>

CRedBlackTreeNode<T>是用于实施CRedBlackTree<T>类的辅助类。

## 描述

CRedBlackTreeNode<T>类是CRedBlackTree<T>的一个节点。在类中实施树状导航方法。

## 声明

```
template<typename T>
class CRedBlackTreeNode
```

## 标题

```
#include <Generic\RedBlackTree.mqh>
```

## 类方法

| 方法 | 描述 |
| --- | --- |
| Value | 返回和设置一个节点值 |
| Parent | 返回和设置一个父节点的指针 |
| Left | 返回和设置一个左节点的指针 |
| Right | 返回和设置一个右节点的指针 |
| Color | 返回和设置一个节点颜色 |
| IsLeaf | 确定指定节点是否为枝叶 |
| CreateEmptyNode | 创建一个新的无父系和子系的黑节点，返回其指针 |

© 2000-2025, MetaQuotes Ltd.

## Value (获得方法)

返回节点值。

```
T Value();
```

返回值

返回节点值。

## Value (设置方法)

设置节点值

```
void Value(
   T  value     // 节点值
   );
```

参数

*value*

[in] 节点值。

© 2000-2025, MetaQuotes Ltd.

## Parent（获得方法）

返回一个父节点的指针。

```
CRedBlackTreeNode<T>* Parent();
```

返回值

返回一个父节点的指针。

## Parent（设置方法）

设置一个父节点的指针。

```
void Parent(
   CRedBlackTreeNode<T>*  node      //一个父节点的指针
   );
```

参数

*node*

[in] 一个父节点的指针。

© 2000-2025, MetaQuotes Ltd.

## Left (获得方法)

返回左节点的指针。

```
CRedBlackTreeNode<T>* Left();
```

返回值

返回左节点的指针。

## Left (设置方法)

设置左节点的指针。

```
void Left(
   CRedBlackTreeNode<T>*  node      // 左节点的指针
   );
```

参数

*node*

[in] 左节点的指针。

© 2000-2025, MetaQuotes Ltd.

## Right (获得方法)

返回一个右节点的指针。

```
CRedBlackTreeNode<T>* Right();
```

返回值

返回一个右节点的指针。

## Right (设置方法)

设置一个右节点的指针。

```
void Right(
   CRedBlackTreeNode<T>*  node     // 一个右节点的指针
   );
```

参数

*node*

[in] 一个右节点的指针。

© 2000-2025, MetaQuotes Ltd.

## Color (获得方法)

返回一个节点颜色。

```
ENUM_RED_BLACK_TREE_NODE_TYPE Color();
```

返回值

返回一个节点颜色。

## Color (设置方法)

设置节点颜色。

```
void Color(
   ENUM_RED_BLACK_TREE_NODE_TYPE  clr     //节点颜色
   );
```

参数

*clr*

[in] 节点颜色。

注意

这个节点颜色通过ENUM_RED_BLACK_TREE_NODE_TYPE值来设置。它可以是两种类型：

- RED_BLACK_TREE_NODE_RED —红色节点；
- RED_BLACK_TREE_NODE_BLACK —黑色节点。

© 2000-2025, MetaQuotes Ltd.

# IsLeaf

确定指定节点是否为枝叶。

```
bool IsLeaf();
```

返回值

如果节点为枝叶，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CreateEmptyNode

创建一个新的无父系和子系的黑节点，返回其指针。

```
static CRedBlackTreeNode<T>* CreateEmptyNode();
```

返回值

返回新节点的指针。

© 2000-2025, MetaQuotes Ltd.

# CLinkedListNode<T>

CLinkedListNode<T>是用于实施CLinkedListNode<T>类的辅助类。

## 描述

CLinkedListNode<T>类是双链接列表CLinkedListNode<T>的节点。在类中实施列表导航方法。

## 声明

```
template<typename T>
class CLinkedListNode
```

## 标题

```
#include <Generic\LinkedList.mqh>
```

## 类方法

| 方法 | 描述 |
| --- | --- |
| List | 返回和设置CLinkedList<T>指针 |
| Next | 返回和设置下一个节点的指针 |
| Previous | 返回和设置上一个节点的指针 |
| Value | 返回和设置节点值 |

© 2000-2025, MetaQuotes Ltd.

## List (获得方法)

返回CLinkedList<T>指针。

```
CLinkedList<T>* List();
```

返回值

返回CLinkedList<T>链接列表指针。

## List (设置方法)

设置CLinkedList<T>指针。

```
void List(
   CLinkedList<T>*  value     //列表指针
   );
```

参数

*value*

[in] 链接列表CLinkedList<T>指针。

© 2000-2025, MetaQuotes Ltd.

## Next (获得方法)

返回下一个节点的指针。

```
CLinkedListNode<T>* Next();
```

返回值

返回下一个节点的指针。

## Next (设置方法)

设置下一个节点的指针

```
void Next(
   CLinkedListNode<T>*  value     // 下一个节点的指针
   );
```

参数

*value*

[in] 下一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

## Previous (获得方法)

返回上一个节点的指针。

```
CLinkedListNode<T>* Previous();
```

返回值

返回上一个节点的指针。

## Previous (设置方法)

设置上一个节点的指针。

```
void Previous(
   CLinkedListNode<T>*  value     //上一个节点的指针
   );
```

参数

*value*

[in] 上一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

## Value (获得方法)

返回节点值。

```
T Value();
```

返回值

返回节点值。

## Value (设置方法)

设置节点值

```
void Value(
   T  value     // 节点值
   );
```

参数

*value*

[in] 节点值。

© 2000-2025, MetaQuotes Ltd.

# CKeyValuePair<TKey, TValue>

CKeyValuePair<TKey,TValue>类实施键/值对。

## 描述

CKeyValuePair<TKey,TValue>类实施处理键/值对中的键和值的方法。

## 声明

```
template<typename TKey, typename TValue>
class CKeyValuePair : public IComparable<CKeyValuePair<TKey,TValue>*>
```

## 标题

```
#include <Generic\HashMap.mqh>
```

## 继承体系

IEqualityComparable
  IComparable
    CKeyValuePair

## 类方法

| 方法 | 描述 |
| --- | --- |
| Key | 获得和设置键/值对的键 |
| Value | 获得和设置键/值对的值 |
| Clone | 创建键和值等于当前键和值的新键/值对 |
| Compare | 比较当前键/值对和指定键/值对 |
| Equals | 检查当前键/值对与指定键/值对是否相等 |
| HashCode | 基于键/值对计算散列值 |

© 2000-2025, MetaQuotes Ltd.

## Key（获得方法）

获得键/值对的键。

```
TKey Key();
```

返回值

返回密钥。

## Key（设置方法）

设置键/值对的键。

```
void Key(
   TKey  key     //键
   );
```

参数

*key*

[in] 键。

© 2000-2025, MetaQuotes Ltd.

## Value (获得方法)

获得键/值对的值。

```
TValue Value();
```

返回值

返回值。

## Value (设置方法)

设置键/值对的值。

```
void Value(
   TValue  value     // 值
   );
```

参数

*value*

[in] 值

© 2000-2025, MetaQuotes Ltd.

# Clone

创建键和值等于当前键和值的新键/值对。

```
TValue>* Clone();
```

返回值

返回新键/值对。

© 2000-2025, MetaQuotes Ltd.

# Compare

比较当前键/值对和指定键/值对。

```
int Compare(
   CKeyValuePair<TKeyTValue>*  pair     // 比较对
   );
```

参数

*pair*

[in] 比较对。

返回值

返回表示当前和过去键-值对比率的数字：

- 如果结果小于0，表示当前键/值对小于过去键/值对
- 如果结果等于0，表示当前键/值对等于过去键/值对
- 如果结果大于0，表示当前键/值对大于过去键/值对

注意

键/值对是基于它们的键来比较。

© 2000-2025, MetaQuotes Ltd.

# Equals

检查当前键/值对与指定键/值对是否相等。

```
bool Equals(
   CKeyValuePair<TKeyTValue>*  pair     // 比较对
   );
```

参数

*pair*

[in] 比较对

返回值

如果键/值对相等，返回true；否则返回false。

注意

键/值对是基于它们的键来比较。

© 2000-2025, MetaQuotes Ltd.

# HashCode

基于键/值对计算散列值。

```
int HashCode();
```

返回值

返回散列代码。

注意

键/值对的散列码等于键散列码。

© 2000-2025, MetaQuotes Ltd.

# CArrayList<T>

CArrayList<T>是实施IList<T>接口的通用类。

## 描述

CArrayList<T>类是实施T类型的动态数据列表。这个类提供了使用列表的基本方法，例如通过索引访问元素，搜索和删除元素，排序等。

## 声明

```
template<typename T>
class CArrayList : public IList<T>
```

## 标题

```
#include <Generic\ArrayList.mqh>
```

## 继承体系

ICollection
  IList
    CArrayList

## 类方法

| 方法 | 描述 |
| --- | --- |
| Capacity | 获取和设置列表的当前容量 |
| Count | 返回列表中的元素数量 |
| Contains | 确定列表是否包含指定值的元素 |
| TrimExcess | 设置列表容量为元素的实际数量 |
| TryGetValue | 获取指定索引的列表元素 |
| TrySetValue | 设置指定索引的列表元素值 |
| Add | 添加元素到列表 |
| AddRange | 添加元素集或元素数组到列表 |
| Insert | 将元素插入到指定索引的列表中 |
| InsertRange | 将元素集或元素数组插入到指定索引的列表中 |
| CopyTo | 从指定索引开始复制列表的所有元素到指定数组 |
| BinarySearch | 在升序列表中搜索指定值 |
| IndexOf | 搜索列表中第一次出现的值 |
| LastIndexOf | 搜索列表中最后出现的值 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 描述 |
| --- | --- |
| Clear | 移除集合的所有元素 |
| Remove | 移除列表中第一次出现的指定元素 |
| RemoveAt | 移除列表指定索引的元素 |
| RemoveRange | 移除列表的元素范围 |
| Reverse | 颠倒列表中元素的顺序 |
| Sort | 排序列表元素 |

© 2000-2025, MetaQuotes Ltd.

## Capacity（获得方法）

返回列表当前容量。

```
int Capacity();
```

返回值

返回列表当前容量。

## Capacity（设置方法）

设置列表当前容量。

```
void Capacity(
   const int  capacity     //容量值
   );
```

参数

*capacity*

[in] 新的容量值。

© 2000-2025, MetaQuotes Ltd.

# Count

返回列表中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定列表是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果列表中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrimExcess

设置列表容量为元素的实际数量，从而释放未使用的内存。

```
void TrimExcess();
```

© 2000-2025, MetaQuotes Ltd.

# TryGetValue

获取指定索引的列表元素。

```
bool TryGetValue(
   const int   index,      //索引
   T&          value       //要编写的变量
   );
```

参数

*index*

[in] 您想要获得值的列表元素的索引。

*&value*

[out] 要编写元素值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrySetValue

设置指定索引的列表元素值。

```
bool TrySetValue(
   const int   index,      //索引
   T           value       //元素值
   );
```

参数

*index*

[in] 您想要设定值的列表元素的索引。

*value*

[in] 设置列表元素的值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到列表。

```
bool Add(
   T  value     // 元素的值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# AddRange

添加元素集或元素数组到列表

添加数组的版本。

```
bool AddRange(
   const T&  array[]                // 要添加的数组
   );
```

添加集合的版本。

```
bool AddRange(
   ICollection<T>*  collection      // 要添加的集合
   );
```

参数

*&array[]*

[in] 要添加的数组。

**collection*

[in] 要添加的集合。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Insert

将元素插入到指定索引的列表中。

```
bool Insert(
   const int  index,      //要插入的索引位置
   T          item        //要被插入的值
   );
```

参数

*index*

[in] 要插入的索引位置。

*item*

[in] 要在指定索引位置插入的值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# InsertRange

将元素集或元素数组插入到指定索引的列表中。

插入数组的版本。

```
bool InsertRange(
   const int  index,                // 要插入的索引位置
   const T&   array[]               //要插入的数组
   );
```

插入集合的版本。

```
bool InsertRange(
   const int        index,          // 要插入的索引位置
   ICollection<T>*  collection      // 要插入的集合
   );
```

参数

*index*

[in] 要插入的索引位置。

*&array[]*

[in]要在指定索引位置插入的数组。

**collection*

[in] 要在指定索引位置插入的集合。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制列表的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用来编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写列表元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# BinarySearch

在升序列表中搜索指定值。

这个版本使用实施ICom parable<T>接口的类搜索值的指定范围进行比较元素。

```
int BinarySearch(
   const int       index,        // 索引起始点
   const int       count,        // 搜索范围
   T               item,         // 搜索值
   IComparer<T>*   comparer      // 比较接口
   );
```

这个版本使用实施ICom parable<T>接口的类进行搜索来比较元素。

```
int BinarySearch(
   T               item,         // 搜索值
   IComparer<T>*   comparer      // 比较接口
   );
```

这个版本使用::Com pare全局方法进行搜索来比较元素。

```
int BinarySearch(
   T  item                       // 搜索值
   );
```

参数

*index*

[in]搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

*item*

[in] 已搜索的值。

**comparer*

[in] 接口用于比较元素。

返回值

返回找到元素的索引。如果搜索值未找到，它将返回最接近值的最小元素的索引。

© 2000-2025, MetaQuotes Ltd.

# IndexOf

搜索列表中第一次出现的值。

整个版本搜索整个列表。

```
int IndexOf(
   T  item                     // 搜索值
   );
```

从指定位置搜索到列表结束的版本。

```
int IndexOf(
   T           item,           // 搜索值
   const int   start_index     //索引起始点
   );
```

从指定范围指定位置搜索的版本。

```
int IndexOf(
   T           item,           // 搜索值
   const int   start_index,    // 索引起始点
   const int   count           //搜索范围
   );
```

参数

*item*

[in] 已搜索的值。

*start_index*

[in] 搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

返回值

返回找到的第一个元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# LastIndexOf

搜索列表中最后出现的值。

整个版本搜索整个列表。

```
int LastIndexOf(
   T  item                     // 搜索值
   );
```

从指定位置搜索到列表结束的版本。

```
int LastIndexOf(
   T           item,           // 搜索值
   const int   start_index     //索引起始点
   );
```

从指定范围指定位置搜索的版本。

```
int LastIndexOf(
   T           item,           // 搜索值
   const int   start_index,    //索引起始点
   const int   count           //搜索范围
   );
```

参数

*item*

[in] 已搜索的值。

*start_index*

[in] 搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

返回值

返回最后找到的元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除集合的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除列表中第一次出现的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveAt

移除列表指定索引的元素。

```
bool RemoveAt(
   const int  index     // 索引
   );
```

参数

*index*

[in] 要移除的元素索引。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveRange

移除列表的元素范围。

```
bool RemoveRange(
   const int  start_index,      //索引起始点
   const int  count             //元素数量
   );
```

参数

*start_index*

[in] 开始删除所在的索引起始点。

*count*

[in] 要删除的元素数量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Reverse

颠倒列表中元素的顺序。

处理整个列表的版本。

```
bool Reverse();
```

处理指定列表元素范围的版本。

```
bool Reverse(
   const int  start_index,     // 索引起始点
   const int  count            // 元素数量
   );
```

参数

*start_index*

[in] 索引起始点。

*count*

[in] 参与操作的列表元素数量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Sort

排序列表元素。

排列列表中所有元素顺序的版本。

```
bool Sort();
```

这个版本使用实施IComparable<T>接口的类排列列表所有元素顺序进行比较元素。

```
bool Sort(
   IComparer<T>*  comparer          //比较接口
   );
```

这个版本使用实施IComparable<T>接口的类排列列表指定元素范围顺序进行比较元素。

```
bool Sort(
   const int  start_index,      //索引起始点
   const int  count             // 元素数量
   IComparer<T>*  comparer      // 比较接口
   );
```

参数

*comparer*

[in] 接口用于比较元素。

*start_index*

[in] 排列顺序开始的索引起始点。

*count*

[in] 排序范围的长度。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CHashMap<TKey, TValue>

CHashMap<TKey, TValue>是实施IMap<TKey, TValue>接口的通用类。

## 描述

CHashMap<TKey, TValue>类可以实施动态散列表，考虑到键的唯一性的要求，其中数据以无序键/值对的形式存储。这个类提供了基本的处理散列表方法，例如访问键值，搜索和删除键/值对等。

## 声明

```
template<typename TKey, typename TValue>
class CHashMap : public IMap<TKey, TValue>
```

## 标题

```
#include <Generic\HashMap.mqh>
```

## 继承体系

ICollection
  IMap
    CHashMap

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加键/值对到散列表 |
| Count | 返回散列表的元素数量 |
| Comparer | 返回IEqualityComparer<T>接口指针，用于组织散列表 |
| Contains | 确定散列表是否包含指定键/值对 |
| ContainsKey | 确定散列表是否包含指定键的键/值对 |
| ContainsValue | CHashMap<TKey, TValue>是实施IMap<TKey, TValue>接口的通用类 |
| CopyTo | 从指定索引开始，复制来自散列表的所有键/值对到指定数组。 |
| Clear | 移除散列表的所有元素 |
| Remove | 移除散列表中第一次出现的键/值对 |
| TryGetValue | 获得散列表中指定键的元素 |
| TrySetValue | 在指定键，更改散列表中键/值对的值 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加键/值对到散列表。

添加生成键/值对的版本。

```
bool Add(
   CKeyValuePair<TKeyTValue>*  pair     //键/值对
   );
```

这个版本在指定键/值基础上添加新键/值对。

```
bool Add(
   TKey    key,                         // 键
   TValue  value                        // 值
   );
```

参数

*pair*

[in] 键/值对。

*key*

[in] 键。

*value*

[in] 值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回散列表的元素数量。

```
int Count();
```

© 2000-2025, MetaQuotes Ltd.

# Comparer

返回IEqualityComparer<T>接口指针，用于组织散列表。

```
IEqualityComparer<TKey>* Comparer() const;
```

返回值

返回IEqualityComparer<T>接口的指针。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定散列表是否包含指定键/值对。

处理生成键/值对的版本。

```
bool Contains(
   CKeyValuePair<TKeyTValue>*  item     // 键/值对
   );
```

这个版本以独立设定键和值的形式处理键/值对。

```
bool Contains(
   TKey   key,                          // 键
   TValue value                         // 值
   );
```

参数

*item*

[in] 键/值对。

*key*

[in] 键。

*value*

[in] 值。

返回值

如果散列表包含指定键和值的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ContainsKey

确定散列表是否包含指定键的键/值对。

```
bool ContainsKey(
   TKey  key     // 键
   );
```

参数

*key*

[in] 键。

返回值

如果散列表包含指定键的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ContainsValue

确定散列表是否包含指定值的键/值对。

```
bool ContainsValue(
   TValue  value     //值
   );
```

参数

*value*

[in] 值。

返回值

如果散列表包含指定值的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始，复制来自散列表的所有键/值对到指定数组。

复制散列表到键/值对数组的版本。

```
int CopyTo(
   CKeyValuePair<TKeyTValue>*&  dst_array[],     // 用于编写键/值对的数组
   const int                    dst_start=0      // 用于编写的索引起始点
   );
```

这个版本复制散列表来为键和值分隔数组。

```
int CopyTo(
   TKey&       dst_keys[],                      // 用来编写键的数组
   TValue&     dst_values[],                    // 用于编写值的数组
   const int   dst_start=0                      // 用于编写的索引起始点
   );
```

参数

*&dst_array[]

[out] 将要编写散列表所有组对的数组。

&dst_keys[]

[out] 将要编写散列表所有键的数组。

&dst_values[]

[out] 将要编写散列表所有值的数组。

dst_start=0

[in] 开始复制的数组索引。

返回值

返回已复制键/值对的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除散列表的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除散列表中第一次出现的键/值对。

这个版本基于生成的键值对移除键值对。

```
bool Remove(
   CKeyValuePair<TKeyTValue>*  item     // 键/值对"
   );
```

这个版本基于键移除键-值对。

```
bool Remove(
   TKey  key                           // 键
   );
```

参数

*item*

[in] 键/值对。

*key*

[in] 键。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TryGetValue

获得散列表中指定键的元素。

```
bool TryGetValue(
   TKey      key,       // 键
   TValue&   value      // 用来编写值的变量
   );
```

参数

*key*

[in] 键。

*&value*

[out] 将要编写键/值对指定值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrySetValue

在指定键，更改散列表中键/值对的值。

```
bool TrySetValue(
   TKey    key,       // 键
   TValue  value      //新值
   );
```

参数

*key*

[in] 键。

*value*

[in] 分配给指定键-值对的新值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CHashSet<T>

CHashSet<T>是实施 ISet<T>接口的通用类。

## 描述

CHashSet<T>类是实施类型T的无序动态数据集， 每个值都是独一无二的。这个类提供了处理集合和相关操作的基本方法，例如：集合的联合和交集，严格和非严格子集的定义等。

## 声明

```
template<typename T>
class CHashSet : public ISet<T>
```

## 标题

```
#include <Generic\HashSet.mqh>
```

## 继承体系

ICollection
  ISet
    CHashSet

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到集合 |
| Count | 返回集合中的元素数量 |
| Comparer | 确定集合是否包含指定值的元素 |
| Contains | 返回IEqualityComparer<T>接口指针，用于组织一个集合 |
| TrimExcess | 设置集合容量为元素的实际数量，从而释放未使用的内存 |
| CopyTo | 从指定索引开始复制集合的所有元素到指定数组 |
| Clear | 移除集合的所有元素 |
| Remove | 移除集合的指定元素 |
| ExceptWith | 产生当前集合和过去集合之间的操作差异（数组） |
| IntersectWith | 产生当前集合和过去集合的操作交集（数组） |
| SymmetricExceptWith | 产生当前集合和过去集合之间的操作对称差（数组） |
| UnionWith | 产生当前集合和过去集合的结合（数组） |
| IsProperSubsetOf | 确定当前集合是否为指定集合或数组的适当子集 |
| IsProperSupersetOf | 确定当前集合是否为指定集合或数组的适当超集合 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 描述 |
| --- | --- |
| IsSubsetOf | 确定当前集合是否为指定集合或数组的子集 |
| IsSupersetOf | 确定当前集合是否为指定集合或数组的超集合 |
| Overlaps | 确定当前集合是否与指定集合或数组重复 |
| SetEquals | 确定当前集合是否包含指定集合或数组的所有元素 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到集合。

```
bool Add(
   T  value     // 元素的值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回集合中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定集合是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果集合中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Comparer

返回IEqualityComparer<T>接口指针，用于组织一个集合。

```
IEqualityComparer<T>* Comparer() const;
```

返回值

返回IEqualityComparer<T>接口的指针。

© 2000-2025, MetaQuotes Ltd.

# TrimExcess

设置集合容量为元素的实际数量，从而释放未使用的内存。

```
void TrimExcess();
```

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制集合的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用于编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写集合元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除集合的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除集合的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ExceptWith

产生当前集合和过去集合之间的操作差异（数组）。从当前集合（数组）移除指定集合（数组）中存在的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
void ExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void ExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 当前集合以外的集合。

*&collection[]*

[in] 当前集合以外的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# IntersectWith

产生当前集合和过去集合的操作交集（数组）。更改当前集合到只包含在指定集合中存在的元素（数组）。

这个版本处理了实施 ICollection<T>接口的集合。

```
void IntersectWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void IntersectWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合相交的集合。

*&collection[]*

[in] 与当前集合相交的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# SymmetricExceptWith

产生当前集合和过去集合之间的操作对称差（数组）。更改当前集合到只包含在源对象或指定集合中存在的元素（数组），但不是同时存在两者中。

这个版本处理了实施 ICollection<T>接口的集合。

```
void SymmetricExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void SymmetricExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 产生对称差的集合。

&collection[]

[in] 产生对称差的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# UnionWith

产生当前集合和过去集合的结合（数组）。将从指定集合（数组）缺失的元素添加到当前集合（数组）。

这个版本处理了实施 ICollection<T>接口的集合。

```
void UnionWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void UnionWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合一致的集合。

*&collection[]*

[in] 与当前集合一致的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# IsProperSubsetOf

确定当前集合是否为指定集合或数组的适当子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为适当子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsProperSupersetOf

确定当前集合是否为指定集合或数组的适当超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为适当超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSubsetOf

确定当前集合是否为指定集合或数组的子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSupersetOf

确定当前集合是否为指定集合或数组的超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前集为超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Overlaps

确定当前集合是否与指定集合或数组重复。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool Overlaps(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool Overlaps(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 确定重复的集合。

*&collection[]*

[in] 确定重复的数组。

返回值

如果当前集和集合或数组重复，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# SetEquals

确定当前集合是否包含指定集合或数组的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool SetEquals(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool SetEquals(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 比较元素的集合。

*&collection[]*

[in] 比较元素的数组。

返回值

如果当前集包括指定集合或数组的所有元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CLinkedList<T>

CLinkedList<T>是实施ICollection<T>接口的通用类。

## 描述

CLinkedList<T>类是实施T类型的动态双链接数据列表。这个类提供了处理双链接列表的基本方法，例如添加，删除，搜索元素等。

## 声明

```
template<typename T>
class CLinkedList : public ICollection<T>
```

## 标题

```
#include <Generic\LinkedList.mqh>
```

## 继承体系

ICollection

CLinkedList

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到链接列表 |
| AddAfter | 在链接列表指定节点之后添加一个元素 |
| AddBefore | 在链接列表指定节点之前添加一个元素 |
| AddFirst | 在链接列表开始添加一个元素 |
| AddLast | 在链接列表结尾添加一个元素 |
| Count | 返回链接列表中的元素数量 |
| Head | 返回链接列表第一个节点上的一个指针 |
| First | 返回链接列表第一个节点上的一个指针 |
| Last | 返回链接列表最后一个节点上的一个指针 |
| Contains | 确定链接列表是否包含指定值的元素 |
| CopyTo | 从指定索引开始复制链接列表的所有元素到指定数组 |
| Clear | 移除链接列表的所有元素 |
| Remove | 移除链接列表中第一次出现的指定元素 |
| RemoveFirst | 移除链接列表的第一个元素 |
| RemoveLast | 移除链接列表的最后一个元素 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 描述 |
| --- | --- |
| Find | 搜索链接列表中第一次出现的指定值 |
| FindLast | 搜索链接列表中最后出现的指定值 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到链接列表。

```
bool Add(
   T  value     // 元素值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# AddAfter

在链接列表指定节点之后添加一个元素

这个版本为按值添加元素。

```
CLinkedListNode<T>* AddAfter(
   CLinkedListNode<T>*   node,          //在该节点之后添加元素
   T                     value          //要添加的元素
   );
```

返回值

返回要添加节点的指针。

这个版本为按值添加元素作为已形成的节点。

```
bool AddAfter(
   CLinkedListNode<T>*   node,          //在该节点之后添加元素
   CLinkedListNode<T>*   new_node       //  要添加的节点
   );
```

参数

*node*

[in]链接列表的节点，在其后将要添加新元素。

*value*

[in]将要添加的元素。

**new_node*

[in]将要添加的节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# AddBefore

在链接列表指定节点之前添加一个元素。

这个版本为按值添加元素。

```
CLinkedListNode<T>* AddBefore(
   CLinkedListNode<T>*   node,         // 在该节点之前添加元素
   T                     value         //要添加的元素
   );
```

返回值

返回要添加节点的指针。

这个版本为按值添加元素作为已形成的节点。

```
bool AddBefore(
   CLinkedListNode<T>*   node,         // 在该节点之前添加元素
   CLinkedListNode<T>*   new_node      // 要添加的节点
   );
```

参数

*node*

[in]链接列表的节点，在其之前将要添加新元素。

*value*

[in]将要添加的元素。

**new_node*

[in]将要添加的节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# AddFirst

在链接列表开始添加一个元素。

这个版本为按值添加元素。

```
CLinkedListNode<T>* AddFirst(
   T    value                     // 要添加的元素
   );
```

返回值

返回要添加节点的指针。

这个版本为按值添加元素作为已形成的节点。

```
bool AddFirst(
   CLinkedListNode<T>*  node      //要添加的节点
   );
```

参数

*value*

[in] 将要添加的元素。

**node*

[in] 将要添加的节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# AddLast

在链接列表结尾添加一个元素

这个版本为按值添加元素。

```
CLinkedListNode<T>* AddLast(
   T  value                        // 要添加的元素
   );
```

返回值

返回要添加节点的指针。

这个版本为按值添加元素作为已形成的节点。

```
bool AddLast(
   CLinkedListNode<T>*  node       // 要添加的节点
   );
```

参数

*value*

[in] 将要添加的元素。

**node*

[in] 将要添加的节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回链接列表中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Head

返回链接列表第一个节点上的一个指针。

```
CLinkedListNode<T>* Head();
```

返回值

返回第一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

# First

返回链接列表第一个节点上的一个指针。

```
CLinkedListNode<T>* First();
```

返回值

返回第一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

# Last

返回链接列表最后一个节点上的一个指针。

```
CLinkedListNode<T>* Last();
```

返回值

返回最后一个节点的指针。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定链接列表是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果链接列表中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制链接列表的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用来编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写链接列表元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除集合的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除链接列表中第一次出现的指定元素

这个版本为按值移除元素。

```
bool Remove(
   T  item                          // 元素值
   );
```

这个版本为按节点指针移除元素。

```
bool Remove(
   CLinkedListNode<T>*  node      // 元素节点
   );
```

参数

*item*

[in] 要被删除的元素值。

**node*

[in] 要被删除的元素节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveFirst

移除链接列表的第一个元素。

```
bool RemoveFirst();
```

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveLast

移除链接列表的最后一个元素。

```
bool RemoveLast();
```

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Find

搜索链接列表中第一次出现的指定值。

```
CLinkedListNode<T>* Find(
   T  value     // 搜索值
   );
```

参数

*value*

[in] 已搜索的值。

返回值

返回第一个找到的包括搜索值的节点指针为success，否则NULL。

© 2000-2025, MetaQuotes Ltd.

# FindLast

搜索链接列表中最后出现的指定值。

```
CLinkedListNode<T>* FindLast(
   T  value     // 搜索值
   );
```

参数

*value*

[in] 已搜索的值。

返回值

返回最后找到的包括搜索值的节点指针为success，否则NULL。

© 2000-2025, MetaQuotes Ltd.

# CQueue<T>

CQueue<T>是实施ICollection<T>接口的通用类。

## 描述

CQueue<T>类是T类型数据的动态集合，并整理为按照FIFO（先进，先出）规则操作的队列。

## 声明

```
template<typename T>
class CQueue : public ICollection<T>
```

## 标题

```
#include <Generic\Queue.mqh>
```

## 继承体系

ICollection
  CQueue

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到队列 |
| Enqueue | 添加元素到队列 |
| Count | 返回队列中的元素数量 |
| Contains | 确定队列是否包含指定值的元素 |
| TrimExcess | 设置队列容量为元素的实际数量，从而释放未使用的内存 |
| CopyTo | 从指定索引开始复制队列的所有元素到指定数组 |
| Clear | 移除队列的所有元素 |
| Remove | 移除队列中第一次出现的指定元素 |
| Dequeue | 返回开始元素并从队列中移除 |
| Peek | 返回开始元素但不从队列中移除 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到队列。

```
bool Add(
   T  value     // 元素值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Enqueue

添加元素到队列。

```
bool Enqueue(
   T  value     // 要添加的元素
   );
```

参数

*value*

[in] 将要添加的元素。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回队列中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定队列是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果队列中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrimExcess

设置队列容量为元素的实际数量，从而释放未使用的内存。

```
void TrimExcess();
```

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制队列的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用来编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写队列元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除队列的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除队列中第一次出现的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Dequeue

返回第一个元素并从队列中移除。

```
T Dequeue();
```

返回值

返回开始元素。

© 2000-2025, MetaQuotes Ltd.

# Peek

返回第一个元素但不从队列中移除。

```
T Peek();
```

返回值

返回开始元素。

© 2000-2025, MetaQuotes Ltd.

# CRedBlackTree<T>

CRedBlackTree<T>是实施ICollection<T>接口的通用类。

## 描述

CRedBlackTree<T>类是实施节点存储T类型数据的动态红黑树。 这个类提供了处理红黑树的基本方法，例如添加，删除，搜索最大和最小值等。

## 声明

```
template<typename T>
class CRedBlackTree : public ICollection<T>
```

## 标题

```
#include <Generic\RedBlackTree.mqh>
```

## 继承体系

ICollection

CRedBlackTree

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到红黑树 |
| Root | 返回红黑树根的一个指针 |
| Count | 返回红黑树中的元素数量 |
| Contains | 确定红黑树是否包含指定值的元素 |
| Comparer | 返回ICom parer<T>接口指针，用于组织红黑树 |
| TryGetMin | 获得红黑树最小元素 |
| TryGetMax | 获得红黑树最大元素 |
| CopyTo | 从指定索引开始复制红黑树的所有元素到指定数组 |
| Clear | 移除链接红黑树的所有元素 |
| Remove | 移除红黑树中出现的指定元素 |
| RemoveMin | 移除红黑树中最小值的元素 |
| RemoveMax | 移除红黑树中最大值的元素 |
| Find | 搜索红黑树中出现的指定值 |
| FindMax | 搜索红黑树中最大值的元素 |
| FindMin | 搜索红黑树中最小值的元素 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到红黑树。

```
bool Add(
   T  value      // 要添加的元素
   );
```

参数

*value*

[in] 将要添加的元素。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回红黑树中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Root

返回红黑树根的一个指针。

```
CRedBlackTreeNode<T>* Root();
```

返回值

返回根指针。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定红黑树是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果红黑树中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Comparer

返回ICom parer<T>接口指针，用于组织红黑树。

```
IComparer<T>* Comparer() const;
```

返回值

返回ICom parer<T>接口的指针。

© 2000-2025, MetaQuotes Ltd.

# TryGetMin

获得红黑树最小元素。

```
bool TryGetMin(
   T&  min     // 用来编写值的变量
   );
```

参数

*&min*

[out] 将要编写最小值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TryGetMax

获得红黑树最大元素。

```
bool TryGetMax(
   T&  max     // 用来编写值的变量
   );
```

参数

*&max*

[out] 将要编写最大值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制红黑树的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用于编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写红黑树元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除链接红黑树的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除红黑树中出现的指定元素。

这个版本移除指定值元素。

```
bool Remove(
   T  value                          // 元素值
   );
```

这个版本为按节点指针移除元素。

```
bool Remove(
   CRedBlackTreeNode<T>*  node      //元素节点
   );
```

参数

*item*

[in]要被删除的元素值。

**node*

[in]要被删除的元素节点。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveMin

移除红黑树中最小值的元素。

```
bool RemoveMin();
```

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# RemoveMax

移除红黑树中最大值的元素。

```
bool RemoveMax();
```

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Find

搜索红黑树中出现的指定值。

```
CRedBlackTreeNode<T>* Find(
   T  value     //搜索值
   );
```

参数

*value*

[in] 已搜索的值。

返回值

返回找到的包括搜索值的节点指针为success，否则NULL。

© 2000-2025, MetaQuotes Ltd.

# FindMin

搜索红黑树中最小值的元素。

```
CRedBlackTreeNode<T>* FindMin();
```

返回值

返回包括最小值的节点指针为success，否则NULL。

© 2000-2025, MetaQuotes Ltd.

# FindMax

搜索红黑树中最大值的元素。

```
CRedBlackTreeNode<T>* FindMax();
```

返回值

返回包括最大值的节点指针为success，否则NULL。

© 2000-2025, MetaQuotes Ltd.

# CSortedMap<TKey, TValue>

CSortedMap<TKey,TValue>是实施IMap<TKey,TValue>接口的通用类。

## 描述

CSortedMap<TKey,TValue>类可以实施动态散列表，考虑到键的唯一性的要求，其中数据存储为按键排序的键/值对。这个类提供了基本的处理散列表方法，例如访问键值，搜索和删除键/值对等。

## 声明

```
template<typename TKey, typename TValue>
class CSortedMap : public IMap<TKey, TValue>
```

## 标题

```
#include <Generic\SortedMap.mqh>
```

## 继承体系

ICollection
  IMap
    CSortedMap

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加键/值对到散列表 |
| Count | 返回已排序散列表中的元素数量 |
| Contains | 确定已排序散列表是否包含指定键/值表 |
| ContainsKey | 确定已排序散列表是否包含指定键的键/值表 |
| ContainsValue | 确定已排序散列表是否包含指定值的键/值表 |
| Comparer | 返回IComparer<T>接口指针，用于组织已排序的散列表 |
| CopyTo | 从指定索引开始，复制来自已排序的散列表的所有键/值对到指定数组 |
| Clear | 移除已排序的散列表的所有元素 |
| Remove | 移除已排序的散列表中第一次出现的键/值对 |
| TryGetValue | 获得已排序的散列表中指定键的元素 |
| TrySetValue | 获得已排序的散列表中指定键的键/值对 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加键/值对到散列表。

添加生成键/值对的版本。

```
bool Add(
   CKeyValuePair<TKeyTValue>*  pair     //键/值对
   );
```

这个版本在指定键/值基础上添加新键/值对。

```
bool Add(
   TKey    key,                         // 键
   TValue  value                        // 值
   );
```

参数

*pair*

[in] 键/值对。

*key*

[in] 键。

*value*

[in] 值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回已排序散列表中的元素数量。

```
int Count();
```

© 2000-2025, MetaQuotes Ltd.

# Comparer

返回IComparer<T>接口指针，用于组织已排序的散列表。

```
IComparer<TKey>* Comparer() const;
```

返回值

返回IComparer<T>接口的指针。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定已排序散列表是否包含指定键/值表。

处理生成键/值对的版本。

```
bool Contains(
   CKeyValuePair<TKeyTValue>*  item     // 键/值对
   );
```

这个版本以独立设定键和值的形式处理键/值对。

```
bool Contains(
   TKey    key,                         // 键
   TValue  value                        // 值
   );
```

参数

*item*

[in] 键/值对。

*key*

[in] 键。

*value*

[in] 值。

返回值

如果已排序的散列表包含指定键和值的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ContainsKey

确定已排序散列表是否包含指定键的键/值表。

```
bool ContainsKey(
   TKey  key     //键
   );
```

参数

*key*

[in] 键。

返回值

如果已排序的散列表包含指定键的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ContainsValue

确定已排序散列表是否包含指定值的键/值表。

```
bool ContainsValue(
   TValue  value     // 值
   );
```

参数

*value*

[in] 值。

返回值

如果已排序的散列表包含指定值的键/值对，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始，复制来自已排序的散列表的所有键/值对到指定数组。

复制散列表到键/值对数组的版本。

```
int CopyTo(
   CKeyValuePair<TKeyTValue>*&  dst_array[],      //用于编写键/值对的数组
   const int                    dst_start=0       //  用于编写的索引起始点
   );
```

这个版本复制散列表来为键和值分隔数组。

```
int CopyTo(
   TKey&       dst_keys[],                       //  用于编写键的数组
   TValue&     dst_values[],                     //用于编写值的数组
   const int   dst_start=0                       //用于编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写散列表所有组对的数组。

*&dst_keys[]*

[out] 将要编写散列表所有键的数组。

*&dst_values[]*

[out] 将要编写散列表所有值的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制键/值对的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除已排序的散列表的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除已排序的散列表中第一次出现的键/值对。

这个版本基于生成的键值对移除键值对。

```
bool Remove(
   CKeyValuePair<TKeyTValue>*  item     // 键/值对
   );
```

这个版本基于键移除键-值对。

```
bool Remove(
   TKey  key                           // 键
   );
```

参数

*item*

[in] 键/值对。

*key*

[in] 键。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TryGetValue

获得已排序的散列表中指定键的元素。

```
bool TryGetValue(
   TKey      key,       // 键
   TValue&   value      // 编写值的变量
   );
```

参数

*key*

[in] 键。

*&value*

[out] 将要编写键/值对指定值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrySetValue

在指定键，更改已排序的散列表中键/值对的值。

```
bool TrySetValue(
   TKey    key,       // 键
   TValue  value      // 新值
   );
```

参数

*key*

[in] 键。

*value*

[in] 分配给指定键-值对的新值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CSortedSet<T>

CSortedSet<T>是实施 ISet<T>接口的通用类。

## 描述

CSortedSet<T>类是实施类型T的已排序动态数据集，每个值都是独一无二的。这个类提供了处理集合和相关操作的基本方法，例如：集合的联合和交集，严格和非严格子集的定义等。

## 声明

```
template<typename T>
class CSortedSet : public ISet<T>
```

## 标题

```
#include <Generic\SortedSet.mqh>
```

## 继承体系

ICollection
  ISet
    CSortedSet

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到已排序集合 |
| Count | 返回已排序集中的元素数量 |
| Contains | 确定已排序集合是否包含指定值的元素 |
| Comparer | 返回ICom parer<T>接口指针，用于组织已排序的集合 |
| TryGetMin | 获得已排序集合的最小元素 |
| TryGetMax | 获得已排序集合的最大元素 |
| CopyTo | 从指定索引开始复制已排序集合的所有元素到指定数组 |
| Clear | 移除已排序集合的所有元素 |
| Remove | 移除已排序集合中出现的指定元素 |
| ExceptWith | 产生当前集合和过去集合之间的操作差异（数组） |
| IntersectWith | 产生当前集合和过去集合的操作交集（数组） |
| SymmetricExceptWith | 产生当前集合和过去集合之间的操作对称差（数组） |
| UnionWith | 产生当前集合和过去集合的结合（数组） |
| IsProperSubsetOf | 确定当前已排序集合是否为指定集合或数组的适当子集 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 描述 |
| --- | --- |
| IsProperSupersetOf | 确定当前已排序集合是否为指定集合或数组的适当超集合 |
| IsSubsetOf | 确定当前已排序集合是否为指定集合或数组的子集 |
| IsSupersetOf | 确定当前已排序集合是否为指定集合或数组的超集合 |
| Overlaps | 确定当前已排序集合是否与指定集合或数组重复 |
| SetEquals | 确定当前已排序集合是否包含指定集合或数组的所有元素 |
| GetViewBetween | 从当前已排序集合获取根据最小和最大值指定的子集 |
| GetReverse | 获取当前已排序集合的复件，其中所有元素都以倒序排列 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到已排序集合。

```
bool Add(
   T  value     // 元素值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回已排序集中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定已排序集合是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果集合中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Comparer

返回ICom parer<T>接口指针，用于组织已排序的集合。

```
IComparer<T>* Comparer() const;
```

返回值

返回ICom parer<T>接口的指针。

© 2000-2025, MetaQuotes Ltd.

# TryGetMin

获得已排序集合的最小元素。

```
bool TryGetMin(
   T&  min     // 用来编写值的变量
   );
```

参数

*&min*

[out] 将要编写最小值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TryGetMax

获得已排序集合的最大元素。

```
bool TryGetMax(
   T&  max     // 用来编写值的变量
   );
```

参数

*&max*

[out] 将要编写最大值的变量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制已排序集合的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用来编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写集合元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除已排序集合的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除已排序集合中出现的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# ExceptWith

产生当前集合和过去集合之间的操作差异（数组）。从当前集合（数组）移除指定集合（数组）中存在的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
void ExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void ExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 当前已排序集合以外的集合。

*&collection[]*

[in] 当前已排序集合以外的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# IntersectWith

产生当前集合和过去集合的操作交集 (数组)。更改当前集合到只包含在指定集合中存在的元素 (数组)。

这个版本处理了实施 ICollection<T>接口的集合。

```
void IntersectWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void IntersectWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合相交的集合。

*&collection[]*

[in] 与当前集合相交的数组。

注意

这个结果被写入当前集合 (数组)。

© 2000-2025, MetaQuotes Ltd.

# SymmetricExceptWith

产生当前集合和过去集合之间的操作对称差（数组）。更改当前集合到只包含在源对象或指定集合中存在的元素（数组），但不是同时存在两者中。

这个版本处理了实施 ICollection<T>接口的集合。

```
void SymmetricExceptWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void SymmetricExceptWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 产生对称差的集合。

*&collection[]*

[in] 产生对称差的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# UnionWith

产生当前集合和过去集合的结合（数组）。将从指定集合（数组）缺失的元素添加到当前集合（数组）。

这个版本处理了实施 ICollection<T>接口的集合。

```
void UnionWith(
   ICollection<T>*  collection     // 集合
   );
```

处理数组的版本。

```
void UnionWith(
   T&  array[]                     // 数组
   );
```

参数

*collection*

[in] 与当前集合一致的集合。

*&collection[]*

[in] 与当前集合一致的数组。

注意

这个结果被写入当前集合（数组）。

© 2000-2025, MetaQuotes Ltd.

# IsProperSubsetOf

确定当前已排序集合是否为指定集合或数组的适当子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection

[in] 确定关系的集合。

&collection[]

[in] 确定关系的数组。

返回值

如果当前已排序集为适当子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsProperSupersetOf

确定当前已排序集合是否为指定集合或数组的适当超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsProperSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsProperSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前已排序集为适当超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSubsetOf

确定当前已排序集合是否为指定集合或数组的子集。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSubsetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSubsetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前已排序集为子集，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# IsSupersetOf

确定当前已排序集合是否为指定集合或数组的超集合。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool IsSupersetOf(
   ICollection<T>*  collection     // 确定关系的集合
   );
```

处理数组的版本。

```
bool IsSupersetOf(
   T&  array[]                     // 确定关系的数组
   );
```

参数

*collection*

[in] 确定关系的集合。

*&collection[]*

[in] 确定关系的数组。

返回值

如果当前已排序集为超集合，则返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Overlaps

确定当前已排序集合是否与指定集合或数组重复。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool Overlaps(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool Overlaps(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 确定重复的集合。

*&collection[]*

[in] 确定重复的数组。

返回值

如果当前已排序集和集合或数组重复，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# SetEquals

确定当前已排序集合是否包含指定集合或数组的所有元素。

这个版本处理了实施 ICollection<T>接口的集合。

```
bool SetEquals(
   ICollection<T>*  collection     // 比较集合
   );
```

处理数组的版本。

```
bool SetEquals(
   T&  array[]                     // 比较数组
   );
```

参数

*collection*

[in] 比较元素的集合。

*&collection[]*

[in] 比较元素的数组。

返回值

如果当前已排序集包括指定集合或数组的所有元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# GetViewBetween

从当前已排序集合获取根据最小和最大值指定的子集

```
bool GetViewBetween(
   T&  array[],          // 用于编写的数组
   T   lower_value,      // 最小值
   T   upper_value       // 最大值
   );
```

参数

*&array[]*

[out] 用于编写子集的数组。

*lower_value*

[in] 范围的最小值。

*upper_value*

[in] 范围的最大值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# GetReverse

获取当前已排序集合的复件，其中所有元素都以倒序排列。

```
bool GetReverse(
   T&  array[]     // 用于编写的数组
   );
```

参数

*&array[]*

[out]用于编写的数组。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# CStack<T>

CStack<T> 是实施ICollection<T>接口的通用类。

## 描述

CStack<T>类是T类型数据的动态集合，并整理为按照LIFO (后进，先出)规则操作的堆栈。

## 声明

```
template<typename T>
class CStack : public ICollection<T>
```

## 标题

```
#include <Generic\Stack.mqh>
```

## 继承体系

ICollection
  CStack

## 类方法

| 方法 | 描述 |
| --- | --- |
| Add | 添加元素到堆栈 |
| Count | 返回堆栈中的元素数量 |
| Contains | 确定堆栈是否包含指定值的元素 |
| TrimExcess | 设置堆栈容量为元素的实际数量 |
| CopyTo | 从指定索引开始复制堆栈的所有元素到指定数组 |
| Clear | 移除堆栈的所有元素 |
| Remove | 移除堆栈中第一次出现的指定元素 |
| Push | 添加元素到堆栈 |
| Peek | 返回开始元素但不从堆栈中移除 |
| Pop | 返回开始元素并从堆栈中移除 |

© 2000-2025, MetaQuotes Ltd.

# Add

添加元素到堆栈。

```
bool Add(
   T  value     // 元素的值
   );
```

参数

*value*

[in] 要添加的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Count

返回堆栈中的元素数量。

```
int Count();
```

返回值

返回元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Contains

确定堆栈是否包含指定值的元素。

```
bool Contains(
   T  item     // 搜索值
   );
```

参数

*item*

[in] 已搜索的值。

返回值

如果堆栈中找到指定值元素，则返回true；否则返回false。

© 2000-2025, MetaQuotes Ltd.

# TrimExcess

设置堆栈容量为元素的实际数量，从而释放未使用的内存。

```
void TrimExcess();
```

© 2000-2025, MetaQuotes Ltd.

# CopyTo

从指定索引开始复制堆栈的所有元素到指定数组。

```
int CopyTo(
   T&          dst_array[],      // 用于编写的数组
   const int   dst_start=0       // 用来编写的索引起始点
   );
```

参数

*&dst_array[]*

[out] 将要编写堆栈元素的数组。

*dst_start=0*

[in] 开始复制的数组的索引。

返回值

返回已复制元素的数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

移除堆栈的所有元素。

```
void Clear();
```

© 2000-2025, MetaQuotes Ltd.

# Remove

移除堆栈中第一次出现的指定元素。

```
bool Remove(
   T  item     // 元素值
   );
```

参数

*item*

[in] 要被删除的元素值。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Push

添加元素到堆栈。

```
bool Push(
   T  value      // 要添加的元素
   );
```

参数

*value*

[in] 将要添加的元素。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

# Peek

返回开始元素但不从堆栈中移除。

```
T Peek();
```

返回值

返回开始元素。

© 2000-2025, MetaQuotes Ltd.

# Pop

返回开始元素并从堆栈中移除。

```
T Pop();
```

返回值

返回开始元素。

© 2000-2025, MetaQuotes Ltd.

# ArrayBinarySearch

使用IComparable<T>接口，搜索升序一维数组指定值来比较元素。

```
template<typename T>
int ArrayBinarySearch(
   T&              array[],          // 用于搜索的数组
   const int       start_index,      // 索引起始点
   const int       count,            // 搜索范围
   T               value,            // 搜索值
   IComparer<T>*   comparer          // 比较接口
   );
```

参数

*&array[]*

[out] 要搜索的数组。

*value*

[in] 已搜索的值。

**comparer*

[in] 接口用于比较元素。

*start_index*

[in] 搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

返回值

返回找到元素的索引。如果搜索值未找到，它将返回最接近值的最小元素的索引。

© 2000-2025, MetaQuotes Ltd.

# ArrayIndexOf

搜索一维数组中第一次出现的值。

```
template<typename T>
int ArrayIndexOf(
   T&          array[],          // 用于搜索的数组
   T           value,            // 搜索值
   const int   start_index,      //索引起始点
   const int   count             //搜索范围
   );
```

参数

*&array[]*

[out] 要搜索的数组。

*value*

[in] 已搜索的值。

*start_index*

[in] 搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

返回值

返回找到的第一个元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# ArrayLastIndexOf

搜索一维数组中最后出现的值。

```
template<typename T>
int ArrayLastIndexOf(
   T&          array[],          // 用于搜索的数组
   T           value,            // 搜索值
   const int   start_index,      // 索引起始点
   const int   count             //搜索范围
   );
```

参数

*&array[]*

[out] 要搜索的数组。

*value*

[in] 已搜索的值。

*start_index*

[in] 搜索开始的索引起始点。

*count*

[in] 搜索范围长度。

返回值

返回最后找到的元素的索引。如果该值未找到，返回-1。

© 2000-2025, MetaQuotes Ltd.

# ArrayReverse

改变一维数组中的元素序列。

```
template<typename T>
bool ArrayReverse(
   T&          array[],          //源数组
   const int   start_index,      // 索引起始点
   const int   count             // 元素数量
   );
```

参数

*&array[]*

[out] 源数组。

*start_index*

[in] 索引起始点。

*count*

[in] 参与操作的数组元素数量。

返回值

成功返回true，否则返回false。

© 2000-2025, MetaQuotes Ltd.

## Compare

比较两个值的大小，其中一个是否 “大于，小于或等于”另一个值。

这个版本比较两个bool值。

```
int Compare(
   const bool  x,          //第一个值
   const bool  y           //第二个值
   );
```

这个版本比较两个char值。

```
int Compare(
   const char  x,          //第一个值
   const char  y           //第二个值
   );
```

这个版本比较两个uchar值。

```
int Compare(
   const uchar  x,         //第一个值
   const uchar  y          //第二个值
   );
```

这个版本比较两个short值。

```
int Compare(
   const short  x,         //第一个值
   const short  y          //第二个值
   );
```

这个版本比较两个ushort值。

```
int Compare(
   const ushort  x,        //第一个值
   const ushort  y         //第二个值
   );
```

这个版本比较两个color值。

```
int Compare(
   const color  x,         //第一个值
   const color  y          //第二个值
   );
```

这个版本比较两个int值。

```
int Compare(
   const int  x,           // 第一个值
   const int  y            // 第二个值
   );
```

© 2000-2025, MetaQuotes Ltd.

这个版本比较两个uint值。

```
int Compare(
   const uint  x,          //第一个值
   const uint  y           //第二个值
   );
```

这个版本比较两个datetime值。

```
int Compare(
   const datetime  x,      //第一个值
   const datetime  y       //第二个值
   );
```

这个版本比较两个long值。

```
int Compare(
   const long  x,          // 第一个值
   const long  y           //第二个值
   );
```

这个版本比较两个ulong值。

```
int Compare(
   const ulong  x,         //第一个值
   const ulong  y          //第二个值
   );
```

这个版本比较两个float值。

```
int Compare(
   const float  x,         //第一个值
   const float  y          //第二个值
   );
```

这个版本比较两个double值。

```
int Compare(
   const double  x,        //第一个值
   const double  y         //第二个值
   );
```

这个版本比较两个string值。

```
int Compare(
   const string  x,        //第一个值
   const string  y         // 第二个值
   );
```

这个版本比较两个其他类型的值。

```
template<typename T>
int Compare(
```

© 2000-2025, MetaQuotes Ltd.

```
   T  x,                    // 第一个值
   T  y                     //第二个值
);
```

参数

*x*

[in] 第一个值

*y*

[in] 第二个值

返回值

返回表示相比较的两个值比率的数字：

- 如果结果小于0，x则小于y (x<y)
- 如果结果等于0，x则等于y (x=y)
- 如果结果大于0，x则大于y (x>y)

注意

如果T类型是实施 IComparable<T>接口的对象，那么这些对象将根据其比较方法进行比较。在所有其他情况下，都返回0 。

© 2000-2025, MetaQuotes Ltd.

# Equals

比较两个值的相等性。

```
template<typename T>
bool Equals(
   T  x,      //第一个值
   T  y       //第二个值
   );
```

参数

*x*

[in] 第一个值

*y*

[in] 第二个值

返回值

如果对象相等，返回true；否则返回false。

注意

如果T类型是实施IEqualityComparable<T>接口的对象，那么这些对象将根据其平等比较方法进行比较。标准的平等比较用于所有其他情况。

© 2000-2025, MetaQuotes Ltd.

# GetHashCode

计算散列码值。

这个版本处理bool型。

```
int GetHashCode(
   const bool   value          // 值
   );
```

这个版本处理char型。

```
int GetHashCode(
   const char   value          //值
   );
```

这个版本处理uchar型。

```
int GetHashCode(
   const uchar   value         //值
   );
```

这个版本处理short型。

```
int GetHashCode(
   const short   value         // 值
   );
```

这个版本处理ushort型。

```
int GetHashCode(
   const ushort   value        // 值
   );
```

这个版本处理color型。

```
int GetHashCode(
   const color   value         // 值
   );
```

这个版本处理 int型。

```
int GetHashCode(
   const int   value           //值
   );
```

这个版本处理 uint型。

```
int GetHashCode(
   const uint   value          //值
   );
```

这个版本处理datetime型。

```
int GetHashCode(
```

© 2000-2025, MetaQuotes Ltd.

```
   const datetime  value     // 值
   );
```

这个版本处理long型。

```
int GetHashCode(
   const long  value         //值
   );
```

这个版本处理ulong型。

```
int GetHashCode(
   const ulong  value        // 值
   );
```

这个版本处理float型。

```
int GetHashCode(
   const float  value        //值
   );
```

这个版本处理double型。

```
int GetHashCode(
   const double  value       // 值
   );
```

这个版本处理string型。

```
int GetHashCode(
   const string  value       // 值
   );
```

这个版本处理其他类型。

```
template<typename T>
int GetHashCode(
   T  value                  //值
   );
```

参数

*value*

[in] 您想要获得散列代码的值。

返回值

返回散列代码。

注意

如果T类型是实施IEqualityComparable<T>接口的对象，那么散列码将基于HashCode方法来获得。在所有其他情况下，散列码将被计算为值类型名的散列值。

© 2000-2025, MetaQuotes Ltd.

# 文件操作

这章节包含文件操作类的技术细节，以及 MQL5 标准库中相关组件的描述。

文件操作类在开发需要文件输入/输出的应用时可节约大量时间。

MQL5 标准库位于终端工作目录的 Include\Files 文件夹。

| 类 | 描述 |
| --- | --- |
| CFile | 基本文件操作类 |
| CFileBin | 二进制文件操作类 |
| CFileTxt | 文本文件操作类 |

© 2000-2025, MetaQuotes Ltd.

# CFile

CFile 是 CFileBin 和 CFileTxt 文件类的基准类。

## 描述

CFile 类可供其所有子类简便地访问 MQL5 API 函数。

## 声明

```
class CFile: public CObject
```

## 标称库文件

```
#include <Files\File.mqh>
```

## 继承体系

CObject

CFile

直接分支

CFileBin, CFilePipe, CFileTxt

## 类方法

| 属性 | |
|---|---|
| Handle | 获取文件句柄 |
| Filename | 获取文件名称 |
| Flags | 获取文件标志 |
| SetUnicode | 设置/清除 FILE_UNICODE 标志 |
| SetCommon | 设置/清除 FILE_COMMON 标志 |
| 常用文件方法 | |
| Open | 打开文件 |
| Close | 关闭文件 |
| 删除 | 删除文件 |
| IsExist | 检查文件存在 |
| Copy | 拷贝文件 |
| Move | 更名/移动文件 |
| Size | 获取文件大小 |
| Tell | 获取当前文件位置 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| Seek | 设置当前文件位置 |
| Flush | 刷新磁盘数据 |
| IsEnding | 检查文件结束 |
| IsLineEnding | 检查行结束 |
| 常用文件夹方法 | |
| FolderCreate | 创建文件夹 |
| FolderDelete | 删除文件夹 |
| FolderClean | 清除文件夹 |
| 搜索方法 | |
| FileFindFirst | 开始文件搜索 |
| FileFindNext | 继续文件搜索 |
| FileFindClose | 关闭搜索句柄 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Handle

获取打开文件的句柄。

```
int  Handle()
```

返回值

已分配到类实例中的打开文件的句柄。如果无分配文件，则返回 -1。

© 2000-2025, MetaQuotes Ltd.

# FileName

获取打开文件的文件名。

```
string  FileName()
```

返回值

已分配到类实例中的打开文件的文件名。如果无分配文件，则返回 ""。

© 2000-2025, MetaQuotes Ltd.

# Flags

获取打开文件的标志。

```
int  Flags()
```

返回值

已分配到类实例中的打开文件的标志。

© 2000-2025, MetaQuotes Ltd.

# SetUnicode

设置/清除 FILE_UNICODE 标志。

```
void  SetUnicode(
   bool  unicode      // 新标志值
   )
```

参数

*unicode*

[输入] FILE_UNICODE 标志的新值。

注释

字符串操作的结果依据 FILE_UNICODE 标志。如果它为 false，则使用 ANSI代码 (单字节字符)。如果它被设置，则使用 UNICODE 代码 (双字节字符)。如果文件已经打开，标志不可被改变。

© 2000-2025, MetaQuotes Ltd.

# SetCommon

设置/清除 FILE_COMMON 标志。

```
void  SetCommon(
   bool  common      // 新标志值
   )
```

参数

*common*

[输入] FILE_COMMON 标志的新值。

注释

FILE_UNICODE 标志确定当前工作文件夹。如果它为 false，使用本地终端文件夹作为当前工作文件夹。如果它为 true，使用通用文件夹作为当前工作文件夹。如果文件已经打开，标志不可被改变。

© 2000-2025, MetaQuotes Ltd.

# Open

打开指定文件，若成功则将其分配到类实例中。

```
int  Open(
   const string  file_name,        // 文件名称
   int           flags,            // 标志
   short         delimiter=9       // 隔离符
   )
```

参数

*file_name*

[输入] 打开的文件名。

*flags*

[输入] 打开标志。

*delimiter=9*

[输入] CSV 文件的分隔符。

返回值

已打开文件的句柄。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# Close

关闭已分配到类实例的文件。

```
void  Close()
```

© 2000-2025, MetaQuotes Ltd.

## 删除

删除已分配到类实例的文件。

```
void  Delete()
```

## 删除

删除指定文件。

```
void  Delete(
   const string  file_name      // 文件名称
   )
```

参数

*file_name*

[输入] 要删除的文件名。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# IsExist

检查文件存在

```
bool  IsExist(
   const string  file_name      // 文件名称
   )
```

参数

*file_name*

[输入] 要检查的文件名。

返回值

true, 如果文件存在。

© 2000-2025, MetaQuotes Ltd.

# Copy

拷贝文件。

```
bool  Copy(
   const string  src_name,      // 源文件名
   int           src_flag,      // 标志
   const string  dst_name,      // 目标文件名
   int           dst_flags      // 标志
   )
```

参数

*src_name*

[输入] 要拷贝的文件名。

*src_flag*

[输入] 要拷贝的文件标志 (仅使用 FILE_COMMON)。

*dst_name*

[输入] 目的文件名。

*dst_flags*

[输入] 目的文件标志 (仅使用 FILE_REWRITE 和 FILE_COMMON)。

返回值

true 如果成功，false 如果未拷贝。

© 2000-2025, MetaQuotes Ltd.

# Move

更名/移动文件。

```
bool  Move(
   const string  src_name,      // 源文件名
   int           src_flag,      // 标志
   const string  dst_name,      // 目标文件名
   int           dst_flags      // 标志
   )
```

参数

*src_name*

[输入] 移动的文件名称。

*src_flag*

[输入] 要拷贝的文件标志 (仅使用 FILE_COMMON)。

*dst_name*

[输入] 目的文件名。

*dst_flags*

[输入] 目的文件标志 (仅使用 FILE_REWRITE 和 FILE_COMMON)。

返回值

true 如果成功，false 如果未移动。

© 2000-2025, MetaQuotes Ltd.

# Size

获取文件大小字节。

```
ulong  Size()
```

返回值

文件大小字节。如果无分配文件，则返回 ULONG_MAX。

© 2000-2025, MetaQuotes Ltd.

# Tell

获取当前文件位置。

```
ulong  Tell()
```

返回值

当前文件位置。如果无分配文件，则返回 ULONG_MAX。

© 2000-2025, MetaQuotes Ltd.

# Seek

设置当前文件位置。

```
void  Seek(
   long                offset,     // 偏移
   ENUM_FILE_POSITION  origin      // 原点
   )
```

参数

*offset*

[输入] 文件偏移字节 (可为负数)。

*origin*

[输入] 偏移的原点。

返回值

true 如果成功, false 如果文件位置未改变。

© 2000-2025, MetaQuotes Ltd.

# Flush

刷新磁盘上的所有文件输入/输出缓存区数据。

```
void  Flush()
```

© 2000-2025, MetaQuotes Ltd.

# IsEnding

检查文件结束。它在文件进行读操作时使用。

```
bool  IsEnding()
```

返回值

true 如果在读取或寻位操作之后到达文件尾。

© 2000-2025, MetaQuotes Ltd.

# IsLineEnding

检查文件的行结束。它在文件进行读操作时使用。

```
bool  IsLineEnding()
```

返回值

true 如果在 txt 或 csv 文件读取操作之后到达行尾 (CR-LF 字符)。

© 2000-2025, MetaQuotes Ltd.

# FolderCreate

创建新文件夹。

```
bool  FolderCreate(
   const string  folder_name      // 文件夹名称
   )
```

参数

*folder_name*

[输入] 创建的文件夹名称。它包含相对于由 FILE_COMMON 标志 定义的文件夹的路径。

返回值

true 如果成功，false 如果文件夹未能创建。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# FolderDelete

删除指定文件夹。

```
bool  FolderDelete(
   const string  folder_name      // 文件夹名称
   )
```

参数

*folder_name*

[输入] 要删除的文件夹名称。它包含相对于由 FILE_COMMON 标志 定义的文件夹的路径。

返回值

true 如果成功，false 如果文件夹未能删除。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# FolderClean

清除指定文件夹。

```
bool  FolderClean(
   const string  folder_name      // 文件夹名称
   )
```

参数

*folder_name*

[输入] 要删除的文件夹名称。它包含相对于由 FILE_COMMON 标志定义的文件夹的路径。

返回值

true 如果成功，false 如果文件夹未能清理。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# FileFindFirst

使用指定过滤器开始文件搜索。

```
int  FileFindFirst(
   const string  filter,         // 搜索过滤器
   string&       file_name       // 文件名字符串的引用
   )
```

参数

*filter*

[输入] 搜索过滤器。

*file_name*

[输出] 第一个被发现的文件名字符串引用。

返回值

若成功，它返回可用于 FileFindNext 做进一步文件搜索的句柄；或者 INVALID_HANDLE，如果没有任何与文件过滤器相对应的文件。

注释

工作文件夹依据 FILE_COMMON 标志，其定义通过 SetCommon() 方法。

© 2000-2025, MetaQuotes Ltd.

# FileFindNext

继续始自函数 FileFindFirst() 的搜索。

```
bool  FileFindNext(
   int       search_handle,     // 搜索句柄
   string&   file_name          // 下一个被发现的文件名字符串引用
   )
```

参数

*search_handle*

[输入] 搜索句柄，通过 FileFindFirst() 方法返回。

*file_name*

[输如] 成功发现的文件名字符串引用。

返回值

true 如果成功，false - 如果没有与指定过滤器相应的文件。

© 2000-2025, MetaQuotes Ltd.

# FileFindClose

关闭搜索句柄。

```
void  FileFindClose(
   int  search_handle      // 搜索句柄
   )
```

参数

*search_handle*

[输入] 搜索句柄，通过 FileFindFirst() 方法返回。

© 2000-2025, MetaQuotes Ltd.

# CFileBin

CFileBin 类用于简便地访问二进制文件。

## 描述

类 CFileBin 可供访问二进制文件。

## 声明

```
class CFileBin: public CFile
```

## 标称库文件

```
#include <Files\FileBin.mqh>
```

## 继承体系

CObject
  CFile
    CFileBin

## 类方法

| 打开方法 | |
|---|---|
| Open | 打开一个二进制文件 |
| 写入方法 | |
| WriteChar | 写入字符或无符号字符类型的变量 |
| WriteShort | 写入短整数或无符号短整数类型的变量 |
| WriteInteger | 写入整数或无符号整数类型的变量 |
| WriteLong | 写入长整数或无符号长整数类型的变量 |
| WriteFloat | 写入浮点数类型的变量 |
| WriteDouble | 写入双精度类型的变量 |
| WriteString | 写入字符串类型的变量 |
| WriteCharArray | 写入字符或无符号字符类型数组的变量 |
| WriteShortArray | 写入短整数或无符号短整数类型数组的变量 |
| WriteIntegerArray | 写入整数或无符号整数类型数组的变量 |
| WriteLongArray | 写入长整数或无符号长整数类型数组的变量 |
| WriteFloatArray | 写入浮点数类型数组的变量 |
| WriteDoubleArray | 写入双精度类型数组的变量 |
| WriteObject | 写入 CObject 类的继承者实例数据 |

© 2000-2025, MetaQuotes Ltd.

| 打开方法 | |
|---|---|
| 读取方法 | |
| ReadChar | 读取字符或无符号字符类型的变量 |
| ReadShort | 读取短整数或无符号短整数类型的变量 |
| ReadInteger | 读取整数或无符号整数类型的变量 |
| ReadLong | 读取长整数或无符号长整数类型的变量 |
| ReadFloat | 读取浮点数类型的变量 |
| ReadDouble | 读取双精度类型的变量 |
| ReadString | 读取字符串类型的变量 |
| ReadCharArray | 读取字符或无符号字符类型数组的变量 |
| ReadShortArray | 读取短整数或无符号短整数类型数组的变量 |
| ReadIntegerArray | 读取整数或无符号整数类型数组的变量 |
| ReadLongArray | 读取长整数或无符号长整数类型数组的变量 |
| ReadFloatArray | 读取浮点数类型数组的变量 |
| ReadDoubleArray | 读取双精度类型数组的变量 |
| ReadObject | 读取 CObject 类的继承者实例数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CFile

Handle, FileName, Flags, SetUnicode, SetCommon, Open, Close, Delete, Size, Tell, Seek, Flush, IsEnding, IsLineEnding, Delete, IsExist, Copy, Move, FolderCreate, FolderDelete, FolderClean, FileFindFirst, FileFindNext, FileFindClose

© 2000-2025, MetaQuotes Ltd.

# Open

打开指定二进制文件，若成功则将其分配到类实例中。

```
int  Open(
   const string  file_name,      // 文件名称
   int           flags           // 标志
   )
```

参数

*file_name*

[输入] 要打开的文件名称。

*flags*

[输入] 文件打开标志 (此处强制设为 FILE_BIN 标志)。

返回值

已打开文件的句柄。

© 2000-2025, MetaQuotes Ltd.

# WriteChar

写入字符或无符号字符类型的变量至文件。

```
uint  WriteChar(
   char  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteShort

写入短整数或无符号短整数类型变量至文件。

```
uint  WriteShort(
   short  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteInteger

写入整数或无符号整数类型变量至文件。

```
uint  WriteInteger(
   int  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteLong

写入长整数或无符号长整数类型变量至文件。

```
uint  WriteLong(
   long  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteFloat

写入 浮点数 类型变量至文件。

```
uint  WriteFloat(
   float  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteDouble

写入 双精度 类型变量至文件。

```
uint  WriteDouble(
   double  value      // 值
   )
```

参数

*value*

[输入] 写入值。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

## WriteString

写入 字符串 类型变量至文件。

```
uint  WriteString(
   const string  value      // 值
   )
```

参数

*value*

[输入] 写入字符串。

返回值

写入的字节数。

## WriteString

写入 字符串 类型变量至文件。

```
uint  WriteString(
   const string  value,     // 值
   int           size       // 大小
   )
```

参数

*value*

[输入] 写入字符串。

*size*

[输入] 写入的字节数量。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteCharArray

写入字符或无符号字符类型数组的变量至文件。

```
uint  WriteCharArray(
   char&  array[],            // 数组引用
   int    start_item=0,       // 开始元素
   int    items_count=-1      // 元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteShortArray

写入短整数或无符号短整数类型数组的变量至文件。

```
uint  WriteShortArray(
   short&  array[],              // 写入的数组
   int     start_item=0,         // 开始元素
   int     items_count=-1        // 写入元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteIntegerArray

写入整数或无符号整数类型数组的变量至文件。

```
uint  WriteIntegerArray(
   int&  array[],             // 写入的数组
   int   start_item=0,        // 开始元素
   int   items_count=-1       // 写入元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteLongArray

写入长整数或无符号长整数类型数组的变量至文件。

```
uint  WriteLongArray(
   long&  array[],            // 写入的数组
   int    start_item=0,       // 开始元素
   int    items_count=-1      // 写入元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteFloatArray

写入浮点数类型数组的变量至文件。

```
uint  WriteFloatArray(
   float&  array[],              // 写入的数组
   int     start_item=0,         // 开始元素
   int     items_count=-1        // 写入元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteDoubleArray

写入双精度类型数组的变量至文件。

```
uint  WriteDoubleArray(
   double&  array[],             // 写入的数组
   int      start_item=0,        // 开始元素
   int      items_count=-1       // 写入元素的数量
   )
```

参数

*array[]*

[输入] 写入数组。

*start_item=0*

[输入] 起始写入的元素。

*items_count=-1*

[输入] 写入元素数量 (-1 - 写入整个数组)。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# WriteObject

写入 CObject 类的继承者实例数据至文件。

```
bool  WriteObject(
   CObject*  object      // 对象引用
   )
```

参数

*object*

[输入] 写入的 CObject 类继承者实例的引用。

返回值

true 如果成功，false 如果数据未能写入。

© 2000-2025, MetaQuotes Ltd.

# ReadChar

从文件里读取字符或无符号字符类型的变量。

```
bool  ReadChar(
   char&  value      // 目标变量
   )
```

参数

*value*

[输入] 字符类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadShort

从文件里读取 短整数 或 无符号短整数 类型的变量。

```
bool  ReadShort(
   short&  value
   )
```

参数

*value*

[输入] 短整数 或短符号长整数 类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadInteger

从文件里读取整数或无符号整数类型的变量。

```
bool  ReadInteger(
   int&  value      // 目标变量
   )
```

参数

*value*

[输入] 整数或无符号整数 类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadLong

从文件里读取 长整数 长整数 无符号长整数 类型的变量。

```
bool  ReadLong(
   long&  value
   )
```

参数

*value*

[输入] 长整数 或无符号长整数 类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadFloat

从文件里读取浮点类型变量。

```
bool  ReadFloat(
   float&  value      // 目标变量
   )
```

参数

*value*

[输入] 浮点数类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadDouble

从文件里读取 双精度 类型变量。

```
bool  ReadDouble(
   double&  value
   )
```

参数

*value*

[输入] 双精度类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

## ReadString

从文件里读取字符串类型的变量。

```
bool  ReadString(
   string&  value      // 目标字符串
   )
```

参数

*value*

[输入] 字符串类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

## ReadString

从文件里读取字符串类型的变量。

```
bool  ReadString(
   string&  value
   )
```

参数

*value*

[输入] 字符串类型的目标变量。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadCharArray

从文件里读取字符或无符号字符类型数组的变量

```
bool  ReadCharArray(
   char&  array[],              // 目标数组
   int    start_item=0,         // 开始元素
   int    items_count=-1        // 读取元素的数量
   )
```

参数

*array[]*

[输入] 字符或无符号字符类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功, false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadShortArray

从文件里读取 短整数 或 无符号短整数 类型数组的变量。

```
bool  ReadShortArray(
   short&  array[],              // 目标数组
   int     start_item=0,         // 开始元素
   int     items_count=-1        // 读取元素的数量
   )
```

参数

*array[]*

[输入] 短整数 或 无符号短整数 类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功, false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadIntegerArray

从文件里读取整数或无符号整数类型数组的变量。

```
bool  ReadIntegerArray(
   int&  array[],             // 目标数组
   int   start_item=0,        // 开始元素
   int   items_count=-1       // 读取元素的数量
   )
```

参数

*array[]*

[输入] 整数或无符号整数类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadLongArray

从文件里读取 长整数 或 无符号长整数 类型数组的变量。

```
bool  ReadLongArray(
   long&  array[],              // 目标数组
   int    start_item=0,         // 开始元素
   int    items_count=-1        // 读取元素的数量
   )
```

参数

*array[]*

[输入] 长整数或无符号长整数类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功, false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadFloatArray

从文件里读取浮点数类型数组的变量。

```
bool  ReadFloatArray(
   float&  array[],              // 目标数组
   int     start_item=0,         // 开始元素
   int     items_count=-1        // 读取元素的数量
   )
```

参数

*array[]*

[输入] 浮点数类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadDoubleArray

从文件里读取双精度类型数组的变量。

```
bool  ReadDoubleArray(
   double&  array[],             // 目标数组
   int      start_item=0,        // 开始元素
   int      items_count=-1       // 读取元素的数量
   )
```

参数

*array[]*

[输入] 双精度类型的目标数组的引用。

*start_item=0*

[输入] 起始读取的元素。

*items_count=-1*

[输入] 读取元素的数量 (-1 - 读取至文件尾)。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# ReadObject

从文件里读取 CObject 类的继承者实例数据。

```
bool  ReadObject(
   CObject*  object      // 对象引用
   )
```

参数

*object*

[输入] 读取的目标 CObject 类继承者实例的引用。

返回值

true 如果成功，false 如果数据未能读取。

© 2000-2025, MetaQuotes Ltd.

# CFileTxt

CFileTxt类用于简便地访问文本文件。

## 描述

类CFileTxt可供访问文本文件。

## 声明

```
class CFileTxt: public CFile
```

## 标称库文件

```
#include <Files\FileTxt.mqh>
```

## 继承体系

CObject
  CFile
    CFileTxt

## 类方法

| 打开方法 | |
|---|---|
| Open | 打开一个文本文件 |
| 写入方法 | |
| WriteString | 写入字符串类型变量至文件 |
| 读取方法 | |
| ReadString | 从文件里读取字符串类型的变量 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CFile

Handle, FileName, Flags, SetUnicode, SetCommon, Open, Close, Delete, Size, Tell, Seek, Flush, IsEnding, IsLineEnding, Delete, IsExist, Copy, Move, FolderCreate, FolderDelete, FolderClean, FileFindFirst, FileFindNext, FileFindClose

© 2000-2025, MetaQuotes Ltd.

# Open

打开指定文本文件，若成功则将其分配到类实例中。

```
int  Open(
   const string  file_name,      // 文件名称
   int           flags           // 标志
   )
```

参数

*file_name*

[输入] 打开的文件名。

*flags*

[输入] 文件打开标志 (此处强制设为 FILE_TXT 标志)。

返回值

已打开的文件句柄。

© 2000-2025, MetaQuotes Ltd.

# WriteString

写入字符串类型变量至文件。

```
uint  WriteString(
   const string  value      // 写入的字符串
   )
```

参数

*value*

[输入] 写入字符串。

返回值

写入的字节数。

© 2000-2025, MetaQuotes Ltd.

# ReadString

从文件里读取字符串类型的变量。

```
string  ReadString()
```

返回值

已被读取的字符串。

© 2000-2025, MetaQuotes Ltd.

# 字符串操作

这章节包含字符串操作类的技术细节，以及 MQL5 标准库中相关组件的描述。

字符串操作类在开发需要进行文本处理操作的应用时可节约大量时间。

MQL5 标准库位于终端工作目录的 Include\Strings 文件夹。

| 类 | 描述 |
| --- | --- |
| CString | 字符串操作类 |

© 2000-2025, MetaQuotes Ltd.

# CString

CString 类用于简便地访问字符串类型变量。

## 描述

CString 类可供其所有子类简便地访问 MQL5 API 字符串函数。

## 声明

```
class CString: public CObject
```

## 标称库文件

```
#include <Strings\String.mqh>
```

## 继承体系

CObject
  CString

## 类方法

| 数据访问方法 | |
|---|---|
| Str | 获取字符串 |
| Len | 获取字符串长度 |
| Copy | 拷贝字符串 |
| 填充方法 | |
| Fill | 以指定字符填充字符串 |
| 分派 | 分配字符串 |
| Append | 追加字符串 |
| Insert | 插入字符穿 |
| 比较方法 | |
| Compare | 比较字符串 |
| CompareNoCase | 比较字符串时大小写不敏感 |
| 子串方法 | |
| Left | 从字符串左侧获取指定数量的字符 |
| Right | 从字符串右侧获取指定数量的字符 |
| Mid | 从字符串里获取指定数量的字符 |
| 修剪/删除方法 | |

© 2000-2025, MetaQuotes Ltd.

| 数据访问方法 | |
|---|---|
| Trim | 从字符串里删除所有前导和尾随出现的指定字符集。 |
| TrimLeft | 从字符串里删除所有前导出现的指定字符集。 |
| TrimRight | 从字符串里删除所有尾随出现的指定字符集。 |
| Clear | 清除字符串 |
| 转换方法 | |
| ToUpper | 转换为大写字母。 |
| ToLower | 转换为小写字母。 |
| Reverse | 反转字符串 |
| 搜索方法 | |
| Find | 搜索首个匹配的子串 |
| FindRev | 搜索最后匹配的子串 |
| Remove | 从字符串里删除子串 |
| Replace | 替换子串 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type

© 2000-2025, MetaQuotes Ltd.

# Str

获取字符串。

```
string  Str() const;
```

返回值

拷贝字符串。

© 2000-2025, MetaQuotes Ltd.

# Len

获取字符串长度。

```
uint  Len() const;
```

返回值

字符串长度。

© 2000-2025, MetaQuotes Ltd.

## Copy

通过引用拷贝字符串。

```
void  Copy(
   string&  copy      // 引用
   ) const;
```

参数

*copy*

[输入] 拷贝的字符串引用。

## Copy

将字符串拷贝至 CString 类实例。

```
void  Copy(
   CString*  copy      // 对象描述符
   ) const;
```

参数

*copy*

[输入] CString 类对象的描述符。

© 2000-2025, MetaQuotes Ltd.

# Fill

以指定字符填充字符串。

```
bool  Fill(
   short  character      // 字符
   )
```

参数

*character*

[输入] 填充的字符。

返回值

true 如果成功，false 如果字符串未能填充。

© 2000-2025, MetaQuotes Ltd.

## 分派

分配字符串。

```
void  Assign(
   const string  str      // 分配的字符串
   )
```

参数

*str*

[输入] 分配的字符串。

## 分派

分配字符串至 CString 类实例。

```
void  Assign(
   CString*  str      // 对象描述符
   )
```

参数

*str*

[输入] 分配的 CString 类对象的描述符。

© 2000-2025, MetaQuotes Ltd.

## Append

追加字符串。

```
void  Append(
   const string  str      // 追加的字符串
   )
```

参数

*str*

[输入] 追加的字符串。

## Append

追加字符串至 CString 类实例。

```
void  Append(
   CString*  string      // 对象描述符
   )
```

参数

*string*

[输入] 追加的 CString 类对象的描述符。

© 2000-2025, MetaQuotes Ltd.

## Insert

插入字符串至指定位置。

```
uint  Insert(
   uint          pos,      // 位置
   const string  str       // 插入的字符串
   )
```

参数

*pos*

[输入] 插入位置。

*str*

[输入] 插入的字符串。

返回值

返回字符串长度。

## Insert

将字符串插入到 CString 类实例的指定位置。

```
uint  Insert(
   uint      pos,      // 位置
   CString*  str       // 对象描述符
   )
```

参数

*pos*

[输入] 插入位置。

*str*

[输入] 插入的 CString 类对象的描述符。

返回值

返回字符串长度。

© 2000-2025, MetaQuotes Ltd.

## Compare

比较字符串。

```
int  Compare(
   const string  str      // 比较的字符串
   ) const;
```

参数

*str*

[输入] 比较的字符串。

返回值

若字符串相等则返回 0，-1 是字符串小于比较的字符串，1 是字符串大于比较的字符串。

## Compare

一个字符串与 CString 类实例字符串进行比较。

```
int  Compare(
   CString*  str      // 对象描述符
   ) const;
```

参数

*str*

[输入] 比较的 CString 类对象的描述符。

返回值

若字符串相等则返回 0，-1 是字符串小于比较的字符串，1 是字符串大于比较的字符串。

© 2000-2025, MetaQuotes Ltd.

## CompareNoCase

比较字符串时大小写不敏感。

```
int  CompareNoCase(
   const string  str      // 比较的字符串
   ) const;
```

参数

*str*

[输入] 比较的字符串。

返回值

若字符串相等则返回 0，-1 是字符串小于比较的字符串，1 是字符串大于比较的字符串。

## CompareNoCase

一个字符串（大小写不敏感）与 CString 类实例字符串进行比较。

```
int  CompareNoCase(
   CString*  str      // 对象描述符
   ) const;
```

参数

*str*

[输入] 比较的 CString 类对象的描述符。

返回值

若字符串相等则返回 0，-1 是字符串小于比较的字符串，1 是字符串大于比较的字符串。

© 2000-2025, MetaQuotes Ltd.

# Left

从字符串左侧获取指定数量的字符。

```
string  Left(
   uint  count       // 字符数量
   )
```

参数

*count*

[输入] 字符数量。

返回值

结果子串。

© 2000-2025, MetaQuotes Ltd.

# Right

从字符串右侧获取指定数量的字符。

```
string  Right(
   uint  count       // 字符数量
   )
```

参数

*count*

[输入] 字符数量。

返回值

结果子串。

© 2000-2025, MetaQuotes Ltd.

# Mid

从字符串里获取指定数量的字符。

```
string  Mid(
   uint  pos,        // 位置
   uint  count       // 字符数量
   )
```

参数

*pos*

[输入]字符串位置。

*count*

[输入] 字符数量。

返回值

结果子串。

© 2000-2025, MetaQuotes Ltd.

# Trim

从字符串里删除所有前导和尾随出现的指定字符集(和 ' ','\t','\r','\n')。

```
int  Trim(
   const string  targets      // 删除的字符集
   )
```

参数

*targets*

[输入] 删除的字符集。

返回值

删除的字符数量。

例如：

```
//--- 例程 CString::Trim
#include <Strings\String.mqh>
//---
void OnStart()
  {
   CString str;
   //---
   str.Assign("   \t\tABCD\r\n");
   printf("源字符串 '%s'",str.Str());
   //---
   str.Trim("DA-DA-DA");
   printf("结果字符串 '%s'",str.Str());
  }
```

© 2000-2025, MetaQuotes Ltd.

# TrimLeft

从字符串里删除所有前导出现的指定字符集(和 ' ','\t','\r','\n')。

```
int  TrimLeft(
   const string  targets      // 删除的字符集
   )
```

参数

*targets*

[输入] 删除的字符集。

返回值

删除的字符数量。

© 2000-2025, MetaQuotes Ltd.

# TrimRight

从字符串里删除所有尾随出现的指定字符集(和 ' ','\t','\r','\n')。

```
int  TrimRight(
   const string  targets      // 删除的字符集
   )
```

参数

*targets*

[输入] 删除的字符集。

返回值

删除的字符数量。

© 2000-2025, MetaQuotes Ltd.

# Clear

清除字符串。

```
bool  Clear()
```

返回值

true 如果成功，false 如果字符串未被清除。

© 2000-2025, MetaQuotes Ltd.

# ToUpper

转换为大写字母。

```
bool  ToUpper()
```

返回值

true 如果成功，false 如果字符串未被转换。

© 2000-2025, MetaQuotes Ltd.

# ToLower

转换为小写字母。

```
bool  ToLower()
```

返回值

true 如果成功，false 如果字符串未被转换。

© 2000-2025, MetaQuotes Ltd.

# Reverse

反转字符串。

```
void  Reverse()
```

© 2000-2025, MetaQuotes Ltd.

# Find

搜索首个匹配的子串。

```
int  Find(
   uint           start,          // 位置
   const string   substring       // 搜索子串
   ) const;
```

参数

*start*

[输入] 开始搜索的字符串索引值，或是 0 代表从头开始。

*substring*

[输入] 搜索的字串。

返回值

第一个匹配子串的首字母索引值；-1 如果子串未发现。

© 2000-2025, MetaQuotes Ltd.

# FindRev

搜索最后匹配的子串。

```
int  FindRev(
   const string  substring      // 子串
   ) const;
```

参数

*substring*

[输入] 搜索的字串。

返回值

最后一个匹配子串的首字母索引值；-1 如果子串未发现。

© 2000-2025, MetaQuotes Ltd.

# Remove

从字符串里删除子串。

```
uint  Remove(
   const string  substring      // 删除的子串
   )
```

参数

*substring*

[输入] 搜索的字串。

返回值

删除子串的数量。

© 2000-2025, MetaQuotes Ltd.

# Replace

从字符串里替换子串。

```
uint  Replace(
   const string  substring,      // 替换的子串
   const string  newstring       // 新的子串
   )
```

参数

*substring*

[输入] 搜索的字串。

*newstring*

[输入] 替换的子串。

返回值

替换子串的数量。

© 2000-2025, MetaQuotes Ltd.

# 图形对象

这章节包含操作图形对象类的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建自定义程序（脚本，智能程序）时，使用图形对象类，将节省大量时间。

MQL5 标准库（就图形对象而言）位于终端工作目录的 Include\ChartObjects 文件夹。

| 类/组 | 描述 |
| --- | --- |
| CChartObject | 图形对象基准类 |
| Lines | "线条"类组 |
| Channels | "通道"类组 |
| 江恩工具 | "江恩"类组 |
| 菲波纳奇工具 | "菲波纳奇"类组 |
| 埃洛特工具 | "埃洛特"类组 |
| Shapes | "形状"类组 |
| Arrows | "箭头"类组 |
| Controls | "控件"类组 |

© 2000-2025, MetaQuotes Ltd.

# CChartObject

CChartObject是MQL5标准库中图表类型的图形对象的基准类。

## 描述

CChartObject类可供其所有子类简便地访问MQL5 API函数。

## 声明

```
class CChartObject : public CObject
```

## 标称库文件

```
#include <ChartObjects\ChartObject.mqh>
```

## 继承体系

CObject

CChartObject

直接分支

CChartObjectArrow, CChartObjectBitmap, CChartObjectBmpLabel, CChartObjectCycles, CChartObjectElliottWave3, CChartObjectEllipse, CChartObjectFiboArc, CChartObjectFiboFan, CChartObjectFiboTimes, CChartObjectHLine, CChartObjectRectangle, CChartObjectSubChart, CChartObjectText, CChartObjectTrend, CChartObjectTriangle, CChartObjectVLine

## 类方法

| 属性 | |
|---|---|
| ChartId | 获取图形拥有者的图表ID |
| Window | 获取图表为图形的窗口数量 |
| 名称 | 获取/设置图形对象的名称 |
| NumPoints | 获取锚点数 |
| 分派 | |
| Attach | 绑定一个图形图表 |
| SetPoint | 设置锚点 |
| 删除 | |
| 删除 | 删除一个图形图表 |
| Detach | 注销一个图形图表 |
| 平移 | |
| ShiftObject | 对象的相对移动 |
| ShiftPoint | 对象落点的相对移动 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| 对象属性 | |
| Time | 获取/设置对象落点的时间坐标 |
| Price | 获取/设置对象落点的价格坐标 |
| Color | 获取/设置对象的颜色 |
| Style | 获取/设置对象的线形 |
| Width | 获取/设置对象的线宽 |
| BackGround | 获取/设置绘制对象背景的标志 |
| Selected | 获取/设置图形对象的 "selected" 标志 |
| Selectable | 获取/设置对象可选择标志 |
| 描述 | 获取/设置对象的描述 |
| Tooltip | 获取/设置对象的提示 |
| Timeframes | 获取/设置对象的可视性标志掩码 |
| Z_Order | 获取/设置图表点击的优先权 |
| CreateTime | 获取对象创建时间 |
| 水平线对象的属性 | |
| LevelsCount | 获取/设置水平线对象的数量 |
| LevelColor | 获取/设置水平线的颜色 |
| LevelStyle | 获取/设置水平线的线形 |
| LevelWidth | 获取/设置水平线的线宽 |
| LevelValue | 获取/设置水平线的位置 |
| LevelDescription | 获取/设置水平线的描述 |
| 访问 MQL5 API 函数 | |
| GetInteger | 获取对象属性的值。 |
| SetInteger | 设置对象属性。 |
| GetDouble | 获取对象属性的值。 |
| SetDouble | 设置对象属性。 |
| GetString | 获取对象属性的值。 |
| SetString | 设置对象属性。 |
| 输入/输出 | |
| virtual Save | 保存文件的虚方法 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

© 2000-2025, MetaQuotes Ltd.

# ChartId

获取图形对象所属的图表 ID。

```
long  ChartId() const
```

返回值

图形对象位于的图表 Id。如果对象未发现，则返回 -1。

例如：

```
//--- 例程 CChartObject::ChartId
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象的图表标识符
   long chart_id=object.ChartId();
  }
```

© 2000-2025, MetaQuotes Ltd.

# Window

获取图表为图形对象的窗口号码。

```
int  Window() const
```

返回值

图形对象所处的图表窗口号码 (0 - 主窗口)。如果对象未发现，则返回 -1。

例如：

```
//--- 例程 CChartObject::Window
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象窗口
   int window=object.Window();
  }
```

© 2000-2025, MetaQuotes Ltd.

## Name (Get方法)

获取图形对象名称。

```
string  Name() const
```

返回值

已挂载到类实例中的图形对象的名称。如果对象未发现，返回 NULL。

## Name (Set方法)

设置图形对象名称。

```
bool  Name(
   string  name      // 新名称
   )
```

参数

*name*

[输入] 图形对象新名称。

返回值

true - 如果成功，false - 如果您未能改变名称。

例如：

```
//--- 例程 CChartObject::Name
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象名称
   string object_name=object.Name();
   if(object_name!="MyChartObject")
     {
      //--- 设置图表对象名称
      object.Name("MyChartObject");
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# NumPoints

获取图形对象的锚点数量。

```
int  NumPoints() const
```

返回值

已挂载到类实例中的图形对象的锚点数量。如果没有已挂载对象，则返回 0。

例如：

```
//--- 例程 CChartObject::NumPoints
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象锚点计数
   int points=object.NumPoints();
  }
```

© 2000-2025, MetaQuotes Ltd.

# Attach

将一个图形对象挂载至一个类实例。

```
bool  Attach(
   long     chart_id,      // 图表 ID
   string   name,          // 对象名
   int      window,        // 图表窗口
   int      points         // 点数
   )
```

参数

*chart_id*

[输出] 图表标识符。

*name*

[输入] 图形对象名。

*window*

[输入] 图表窗口号码 (0 -主窗口)。

*points*

[输入] 图形对象锚点数。

返回值

true - 如果成功, false - 如果操作失败。

例如：

```
//--- 例程 CChartObject::Attach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 挂载图表对象
   if(!object.Attach(ChartID(),"MyObject",0,2))
     {
      printf("对象挂载出错");
      return;
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# SetPoint

设置图形对象锚点的新坐标。

```
bool  SetPoint(
   int       point,          // 锚点号码
   datetime  new_time,       // 时间坐标
   double    new_price       // 价格坐标
   )
```

参数

*point*

[输入] 锚点号码。

*new_time*

[输入] 指定锚点的新时间坐标。

*new_price*

[输入] 指定锚点的新价格坐标。

返回值

true - 如果成功，false - 如果您未能改变此锚点坐标。

例如：

```
//--- 例程 CChartObject::SetPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   if(object.NumPoints()>0)
     {
      //--- 设置图表对象锚点
      object.SetPoint(0,CurrTime(),price);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# 删除

从指图表里删除一个已挂载的图形对象。

```
bool  Delete()
```

返回值

true - 如果成功，false - 如果删除失败。

例如：

```
//--- 例程 CChartObject::Delete
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 分离图表对象
   if(!object.Delete())
     {
      printf("对象删除错误");
      return;
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# Detach

分离图形对象。

```
void  Detach()
```

返回值

无。

例如：

```
//--- 例程 CChartObject::Detach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 分离图表对象
   object.Detach();
  }
```

© 2000-2025, MetaQuotes Ltd.

# ShiftObject

移动一个图形对象。

```
bool  ShiftObject(
   datetime  d_time,       // 时间坐标增量
   double    d_price       // 价格坐标增量
   )
```

参数

*d_time*

[输入] 所有锚点的时间坐标增量。

*d_price*

[输入] 所有锚点的价格坐标增量。

返回值

true - 如果成功，false - 如果您未能平移对象。

例如：

```
//--- 例程 CChartObject::ShiftObject
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //--- 平移图表对象
   object.ShiftObject(d_time,d_price);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ShiftPoint

平移图形对象的一个指定锚点。

```
bool  ShiftPoint(
   int       point,       // 锚点号码
   datetime  d_time,      // 时间坐标增量
   double    d_price      // 价格坐标增量
   )
```

参数

*point*

[输入] 锚点号码。

*d_time*

[输入] 指定锚点的时间坐标增量。

*d_price*

[输入] 指定锚点的价格坐标增量。

返回值

true - 如果成功，false - 如果您未能平移锚点。

例如：

```
//--- 例程 CChartObject::ShiftPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //---
   if(object.NumPoints()>0)
     {
      //--- 平移图表对象锚点
      object.ShiftPoint(0,d_time,d_price);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Time (Get方法)

获取图形对象指定锚点的时间坐标。

```
datetime  Time(
   int  point      // 锚点号码
   ) const
```

参数

*point*

[输入] 锚点号码。

返回值

已挂载到类实例中的图形对象的指定锚点的时间座标。如果没有挂载对象或对象没有指定锚点，则返回 0。

## Time (Set方法)

设置图形对象指定锚点的时间坐标。

```
bool  Time(
   int       point,        // 锚点号码
   datetime  new_time      // 时间
   )
```

参数

*point*

[输入] 锚点号码。

*new_time*

[输入] 指定锚点的新时间坐标。

返回值

true - 如果成功，false - 如果您未能改变时间坐标。

例如：

```
//--- 例程 CChartObject::Time
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- 获取图表对象锚点时间
      datetime point_time=object.Time(i);
      if(point_time==0)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象锚点时间
          object.Time(i,TimeCurrent());
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## Price（Get方法）

获取图形对象指定锚点的价格坐标。

```
double  Price(
   int  point      // 锚点号码
   ) const
```

参数

*point*

[输入] 锚点号码。

返回值

已挂载到类实例中的图形对象的指定锚点的价格座标。如果没有挂载对象或没有指定锚点，则返回 EMPTY_VALUE。

## Price（Set方法）

设置图形对象指定锚点的价格坐标。

```
bool  Price(
   int     point,          // 锚点号码
   double  new_price       // 价格
   )
```

参数

*point*

[输入] 锚点号码。

*new_price*

[输入] 指定锚点的新价格坐标。

返回值

true - 如果成功，false - 如果您未能改变价格坐标。

例如：

```
//--- 例程 CChartObject::Price
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- 获取图表对象锚点价格
      double point_price=object.Price(i);
```

© 2000-2025, MetaQuotes Ltd.

```
      if(point_price!=price)
        {
         //--- 设置图表对象锚点价格
         object.Price(i,price);
        }
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Color (Get 方法)

获取图形对象的线条颜色。

```
color  Color() const
```

返回值

已挂载到类实例中的图形对象线条颜色。如果此处没有对象加载，则返回 CLR_NONE。

## Color (Set 方法)

设置图形对象的线条颜色。

```
bool  Color(
   color  new_color      // 新颜色
   )
```

参数

*new_color*

[输入] 线条新颜色。

返回值

true - 如果成功，false - 如果您未能改变颜色。

例如：

```
//--- 例程 CChartObject::Color
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象颜色
   color object_color=object.Color();
   if(object_color!=clrRed)
     {
     //--- 设置图表对象颜色
     object.Color(clrRed);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Style (Get 方法)

获取图形对象线条的线形。

```
ENUM_LINE_STYLE  Style() const
```

返回值

已挂载到类实例中的图形对象线条的线形。如果没有已挂载对象，则返回 WRONG_VALUE。

## Style (Set 方法)

设置图形对象线条的线形。

```
bool  Style(
   ENUM_LINE_STYLE  new_style      // 线形
   )
```

参数

*new_style*

[输入] 线形的新值。

返回值

true - 如果成功，false - 如果您未能改变线形。

例如：

```
//--- 例程 CChartObject::Style
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象线形
   ENUM_LINE_STYLE style=object.Style();
   if(style!=STYLE_SOLID)
     {
      //--- 设置图表对象线形
      object.Style(STYLE_SOLID);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Width (Get方法)

获取图形对象的宽度。

```
int  Width() const
```

返回值

已挂载到类实例中的图形对象的线宽。如果没有已挂载对象，则返回 -1。

## Width (Set方法)

设置图形对象线条的线宽。

```
bool  Width(
   int  new_width      // 粗细
   )
```

参数

*new_width*

[输入] 新的线宽值。

返回值

true - 如果成功，false - 如果您未能改变宽度。

例如：

```
//--- 例程 CChartObject::Width
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象宽度
   int width=object.Width();
   if(width!=1)
     {
      //--- 设置图表对象宽度
      object.Width(1);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Background (Get方法)

获取在背景上绘制对象的标志。

```
bool  Background() const
```

返回值

在背景上绘制已加载至类实例中的图形对象的标志。如果没有已挂载对象，则返回 false。

## Background (Set方法)

设置在背景上绘制对象的标志。

```
bool  Background(
   bool  background      // 标志值
   )
```

参数

*background*

[输入] 在背景上绘制对象的新标志值。

返回值

true - 如果成功，false - 如果您未能改变标志。

例如：

```
//--- 例程 CChartObject::Background
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象背景标志
   bool background_flag=object.Background();
   if(!background_flag)
     {
     //--- 设置图表对象背景标志
     object.Background(true);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Selected (Get方法)

获取图形对象被选中的指示标志。换句话说 - 图形对象能否被选中。

```
bool  Selected() const
```

返回值

已加载至类实例中的图形对象状态，被选中。如果没有已挂载对象，则返回 false。

## Selected (Set方法)

设置图形对象被选中的指示标志。

```
bool  Selected(
   bool  selected      // 标志值
   )
```

参数

*selected*

[输入] 新的 标志值。

返回值

true - 如果成功，false - 如果您未能改变标志。

例如：

```
//--- 例程 CChartObject::Selected
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获得图表对象 "selected" 标志
   bool selected_flag=object.Selected();
   if(selected_flag)
     {
     //--- 设置图表对象 "selected" 标志
     object.Selected(false);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Selectable (Get 方法)

获取图形对象有能力被选择的指示标志。换句话说 - 图形对象能否被选择。

```
bool  Selectable() const
```

返回值

已加载至类实例中的图形对象有能力被选择的指示标志。如果没有已挂载对象，则返回 false。

## Selectable (Set 方法)

设置图形对象有能力被选择的指示标志。

```
bool  Selectable(
   bool  selectable      // 标志值
   )
```

参数

*selectable*

[输入] 新的标志值。

返回值

true - 如果成功，false - 如果您未能改变标志。

例如：

```
//--- 例程 CChartObject::Selectable
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象的 "selectable" 标志
   bool selectable_flag=object.Selectable();
   if(selectable_flag)
     {
     //--- 设置图表对象的 "selectable" 标志
     object.Selectable(false);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Description (Get方法)

获取图形对象的描述(文本)。

```
string  Description() const
```

返回值

已挂载到类实例中的图形对象的描述(文本)。如果没有已挂载对象，则返回 NULL。

## Description (Set方法)

设置图形对象的描述(文本)。

```
bool  Description(
   string  text      // 文本
   )
```

参数

*text*

[输入] 新描述 (文本)。

返回值

true - 如果成功，false - 如果您未能改变描述 (文本)。

例如：

```
//--- 例程 CChartObject::Description
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象描述
   string description=object.Description();
   if(description=="")
     {
      //--- 设置图表对象描述
      object.Description("MyObject");
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Tooltip (Get方法)

获取图形对象的提示文本。

```
string  Tooltip() const
```

返回值

已挂载到类实例中的图形对象的提示文本。如果没有已挂载对象，则返回 NULL。

## Tooltip (Set方法)

设置图形对象的提示文本。

```
bool  Tooltip(
   string  new_tooltip      // 新的提示文本
   )
```

参数

*new_tooltip*

[输入] 新的提示文本。

返回值

true - 如果成功，false - 如果修改提示文本失败。

注释：

如果属性未设置，则显示由终端自动生成的提示。提示可通过设置 "\n"(换行)值来禁用。

© 2000-2025, MetaQuotes Ltd.

## Timeframes (Get方法)

获取图形对象的可视性标志。

```
int  Timeframes() const
```

返回值

已挂载到类实例中的图形对象的可视性标志。如果没有已挂载对象，则返回 0。

## Timeframes (Set方法)

设置图形对象的可视性标志。

```
bool  Timeframes(
   int  new_timeframes      // 可视性标志
   )
```

参数

*new_timeframes*

[输入] 图形对象新可视性标志。

返回值

true - 如果成功，false - 如果您未能改变可视性标志。

例如：

```
//--- 例程 CChartObject::Timeframes
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象时间帧
   int timeframes=object.Timeframes();
   if(!(timeframes&OBJ_PERIOD_H1))
     {
      //--- 设置图表对象时间帧
      object.Timeframes(timeframes|OBJ_PERIOD_H1);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

## Z_Order (Get方法)

获取点击图表上图形对象时的优先权 (CHARTEVENT_CLICK)。

```
long  Z_Order() const
```

返回值

已挂载到类实例中的图形对象的优先权。如果此处没有对象加载，则返回 0。

## Z_Order (Set方法)

设置点击图表上图形对象时的优先权 (CHARTEVENT_CLICK)。

```
bool  Z_Order(
   long  value      // 新优先权
   )
```

参数

*value*

[输入] 点击图表上图形对象时的新优先权 (CHARTEVENT_CLICK).

返回值

true - 如果成功，false - 如果您未能改变优先权。

注释

Z_Order 是点击图表上图形对象时的优先权 (CHARTEVENT_CLICK)。通过设置大于 0(省缺) 的值，您可以提升对象优先权。

© 2000-2025, MetaQuotes Ltd.

# CreateTime

获取图形对象的创建时间。

```
datetime  CreateTime() const
```

返回值

已挂载到类实例中的图形对象的创建时间。如果没有已挂载对象，则返回 0。

例如：

```
//--- 例程 CChartObject::CreateTime
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象的创建时间
   datetime create_time=object.CreateTime();
  }
```

© 2000-2025, MetaQuotes Ltd.

## LevelsCount (Get 方法)

获取图形对象的水平线数量。

```
int  LevelsCount() const
```

返回值

已挂载到类实例中的图形对象的水平线数量。如果没有已挂载对象，则返回 0。

## LevelsCount (Set 方法)

设置图形对象的水平线数量。

```
bool  LevelsCount(
   int  levels      // 水平线数量
   )
```

参数

*levels*

[输入] 图形对象的水平线新数量值。

返回值

true - 如果成功，false - 如果您未能改变水平线数量。

例如：

```
//--- 例程 CChartObject::LevelsCount
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取图表对象水平线计数
   int levels_count=object.LevelsCount();
   //--- 设置图表对象水平线计数
   object.LevelsCount(levels_count+1);
  }
```

© 2000-2025, MetaQuotes Ltd.

## LevelColor (Get方法)

获取图形对象指定水平线的颜色。

```
color  LevelColor(
   int  level      // 水平线号码
   ) const
```

参数

*level*

[输入] 水平线号码。

返回值

已挂载到类实例中的图形对象的指定水平线颜色。如果没有挂载对象或没有指定水平线，则返回 CLR_NONE。

## LevelColor (Set方法)

设置图形对象指定水平线的颜色。

```
bool  LevelColor(
   int    level,          // 水平线号码
   color  new_color       // 新颜色
   )
```

参数

*level*

[输入] 水平线号码。

*new_color*

[输入] 指定水平线的新颜色。

返回值

true - 如果成功，false - 如果您未能改变颜色。

例如：

```
//--- 例程 CChartObject::LevelColor
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 获取图表对象水平线颜色
      color level_color=object.LevelColor(i);
      if(level_color!=clrRed)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象水平线颜色
          object.LevelColor(i,clrRed);
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## LevelStyle（Get方法）

获取图形对象指定水平线的线形。

```
ENUM_LINE_STYLE  LevelStyle(
   int  level      // 水平线号码
   ) const
```

参数

*level*

[输入] 水平线号码。

返回值

已挂载到类实例中的图形对象的指定水平线的线形。如果没有挂载对象或对象没有指定水平线，则返回 WRONG_VALUE。

## LevelStyle（Set方法）

设置图形对象指定水平线的线形。

```
int  LevelStyle(
   int              level,     // 水平线号码
   ENUM_LINE_STYLE  style      // 线形
   )
```

参数

*level*

[输入] 水平线号码。

*style*

[输入] 指定水平线的新线形。

返回值

true - 如果成功，false - 如果您未能改变线形。

例如：

```
//--- 例程 CChartObject::LevelStyle
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 获取图表对象水平线线形
      ENUM_LINE_STYLE level_style=object.LevelStyle(i);
      if(level_style!=STYLE_SOLID)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象水平线线形
          object.LevelStyle(i,STYLE_SOLID);
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## LevelWidth (Get方法)

获取图形对象指定水平线的线宽。

```
int  LevelWidth(
   int  level      // 水平线号码
   ) const
```

参数

*level*

[输入] 水平线号码。

返回值

已挂载到类实例中的图形对象指定水平线的线宽。如果没有挂载对象或对象没有指定水平线，则返回 -1。

## LevelWidth (Set方法)

设置图形对象指定水平线的线宽。

```
bool  LevelWidth(
   int  level,          // 水平线号码
   int  new_width       // 新宽度
   )
```

参数

*level*

[输入] 水平线号码。

*new_width*

[输入] 指定水平线的新宽度。

返回值

true - 如果成功，false - 如果您未能改变宽度。

例如：

```
//--- 例程 CChartObject::LevelWidth
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 获取图表对象水平线宽度
      int level_width=object.LevelWidth(i);
      if(level_width!=1)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象水平线宽度
          object.LevelWidth(i,1);
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## LevelValue (Get方法)

获取图形对象指定水平线的值。

```
double  LevelValue(
   int  level      // 水平线号码
   ) const
```

参数

*level*

[输入] 水平线号码。

返回值

已绑定到类实例中的图形对象水平线的值。如果没有绑定对象或对象水平线未指定，则返回EMPTY_VALUE。

## LevelValue (Set方法)

设置图形对象指定水平线的值。

```
bool  LevelValue(
   int     level,          // 水平线号码
   double  new_value       // 新值
   )
```

参数

*level*

[输入] 水平线号码。

*new_value*

[输入] 指定水平线的新值。

返回值

true - 如果成功，false - 如果您未能改变值。

例如：

```
//--- 例程 CChartObject::LevelValue
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 获取图表对象水平线值
      double level_value=object.LevelValue(i);
      if(level_value!=0.1*i)
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象水平线值
          object.LevelValue(i,0.1*i);
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

## LevelDescription (Get方法)

获取图形对象的水平线描述(文本)。

```
string  LevelDescription(
   int  level       // 水平线号码
   ) const
```

参数

*level*

[输入] 图形对象水平线的号码。

返回值

已绑定到类实例中的图形对象的水平线描述(文本)。如果没有绑定对象或对象没有指定水平线，则返回 NULL。

## LevelDescription (Set方法)

设置图形对象的水平线描述(文本)。

```
bool  LevelDescription(
   int     level ,     // 水平线号码
   string  text        // 文本
   )
```

参数

*level*

[输入] 图形对象水平线的号码。

*text*

[输入]图形对象水平线的新描述(文本)值。

返回值

true - 如果成功，false - 如果您未能改变描述 (文本)。

例如：

```
//--- 例程 CChartObject::LevelDescription
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 获取图表对象水平线描述
      string level_description=object.LevelDescription(i);
      if(level_description=="")
```

© 2000-2025, MetaQuotes Ltd.

```
         {
          //--- 设置图表对象水平线描述
          object.LevelDescription(i,"Level_"+IntegerToString(i));
         }
      }
   }
```

© 2000-2025, MetaQuotes Ltd.

# GetInteger

可供简便地访问 MQL5 API ObjectGetInteger() 函数来获取绑定实例的图形类的整数属性值。函数有两个调用版本：

无需正确性验证即可获取一个属性值

```
long  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,          // 整数属性标识符
   int                           modifier=-1       // 限定符
   ) const
```

参数

*prop_id*

[输入] 整数图形属性 ID。

*modifier=-1*

[输入] 限定符（索引）整数特征。

返回值

如果成功，则返回整数类型的属性值，若出错，则返回 0。

验证处治的正确性再获取属性值

```
bool  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,       // 整数属性标识符
   int                           modifier,      // 限定符
   long&                         value          // 链接到变量
   ) const
```

参数

*prop_id*

[输入] 对象的整数图形属性值。

*modifier*

[输入] 限定符（索引）整数特征。

*value*

[输出] 容纳属性整数值的变量引用。

返回值

true - 如果成功，false -如果您未能获取一个整数特征值。

例如：

```
//--- 例程 CChartObject::GetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
  CChartObject object;
  //--- 通过简便方法获取图表颜色
  printf("对象颜色是 %s",ColorToString(object.GetInteger(OBJPROP_COLOR),true));
  //--- 通过经典方法获取图表颜色
  long color_value;
  if(!object.GetInteger(OBJPROP_COLOR,0,color_value))
    {
     printf("获取整数属性出错 %d",GetLastError());
     return;
    }
  else
     printf("对象颜色是 %s",color_value);
  for(int i=0;i<object.LevelsCount();i++)
    {
     //--- 通过简便方法获取水平线宽度
     printf("水平线 %d 宽度是 %d",i,object.GetInteger(OBJPROP_LEVELWIDTH,i));
     //--- 通过经典方法获取水平线宽度
     long width_value;
     if(!object.GetInteger(OBJPROP_LEVELWIDTH,i,width_value))
       {
        printf("获取整数属性出错 %d",GetLastError());
        return;
       }
     else
        printf("水平线 %d 宽度是 %d",i,width_value);
    }
}
```

© 2000-2025, MetaQuotes Ltd.

# SetInteger

可供简便地访问 MQL5 API ObjectSetInteger() 函数来获取绑定实例的图形类的整数属性值。函数有两种调用版本：

无需限定符即可修改一个属性值

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER   prop_id,      // 整数属性标识符
   long                           value         // 新值
   )
```

参数

*prop_id*

[输入] 对象的整数图形属性值。

*value*

[输入] 整数特征的新可变值。

根据限定符设置一个属性值

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER   prop_id,       // 整数属性标识符
   int                            modifier,      // 限定符
   long                           value          // 新值
   )
```

参数

*prop_id*

[输入] 对象的整数图形属性值。

*modifier*

[输入] 限定符（索引）整数特征。

*value*

[输入] 整数特征的新可变值。

返回值

true - 如果成功，false - 如果您未能改变整数特征值。

例如：

```
//--- 例程 CChartObject::SetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 设置图表对象新颜色
   if(!object.SetInteger(OBJPROP_COLOR,clrRed))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       printf("设置整数属性错误 %d",GetLastError());
       return;
      }
   for(int i=0;i<object.LevelsCount();i++)
      {
       //--- 设置水平线宽度
       if(!object.SetInteger(OBJPROP_LEVELWIDTH,i,i))
          {
           printf("设置整数属性错误 %d",GetLastError());
           return;
          }
      }
  }
```

© 2000-2025, MetaQuotes Ltd.

# GetDouble

可供简便地访问 MQL5 API ObjectGetDouble() 函数来获取已分配到类实例中的图形对象的双精度属性值（浮点或双精度类型）。函数有两个调用版本：

无需正确性验证即可获取一个属性值

```
double  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,          // 双精度属性标识符
   int                          modifier=-1      // 限定符
   ) const
```

参数

*prop_id*

[输入] 双精度图形属性 ID。

*modifier=-1*

[输入] 限定符（索引）双精度特征。

返回值

如果成功，则返回双精度类型的属性值，若出错，则返回 EMPTY_VALUE。

验证处治的正确性再获取属性值

```
bool  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,       // 双精度属性标识符
   int                          modifier,      // 限定符
   double&                      value          // 链接到变量
   ) const
```

参数

*prop_id*

[输入] 双精度图形属性 ID。

*modifier*

[输入] 限定符（索引）双精度特征。

*value*

[输出] 容纳属性双精度值的变量引用。

返回值

true - 如果成功，false - 如果您未能获取一个双精度特征值。

例如：

```
//--- 例程 CChartObject::GetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
  CChartObject object;
  //---
  for(int i=0;i<object.LevelsCount();i++)
    {
     //--- 通过简便方法获取值
     printf("水平线 %d 值=%f",i,object.GetDouble(OBJPROP_LEVELVALUE,i));
     //--- 通过经典方法获取值
     double value;
     if(!object.SetDouble(OBJPROP_LEVELVALUE,i,value))
       {
        printf("获取双精度属性出错 %d",GetLastError());
        return;
       }
     else
        printf("水平线 %d 值=%f",i,value);
    }
 }
```

© 2000-2025, MetaQuotes Ltd.

# SetDouble

可供简便地访问 MQL5 API ObjectSetDouble() 函数来修改绑定实例的图形类的双精度属性值(浮点或双精度)。函数有两个调用版本:

无需限定符即可修改一个属性值

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,      // 双精度属性标识符
   double                        value         // Value
   )
```

参数

*prop_id*

[输入] 双精度图形属性 ID。

*value*

[输入] 双精度特征的新可变值。

根据限定符设置一个属性值

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,       // 双精度属性标识符
   int                           modifier,      // 限定符
   double                        value          // 值
   )
```

参数

*prop_id*

[输入] 双精度图形属性 ID。

*modifier*

[输入] 限定符 (索引) 双精度特征。

*value*

[输入] 双精度特征的新可变值。

返回值

true - 如果成功, false - 如果您未能改变双精度特征值。

例如:

```
//--- 例程 CChartObject::SetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
//---
   for(int i=0;i<object.LevelsCount();i++)
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 设置图表对象水平线值
       if(!object.SetDouble(OBJPROP_LEVELVALUE,i,0.1*i))
         {
          printf("设置双精度属性错误 %d",GetLastError());
          return;
         }
      }
  }
```

© 2000-2025, MetaQuotes Ltd.

## GetString

可供简便地访问 MQL5 API ObjectGetString() 函数来获取绑定实例的图形类的字符串属性值。函数有两个调用版本：

无需正确性验证即可获取一个属性值

```
string  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,          // 字符串属性标识符
   int                          modifier=-1      // 限定符
   ) const
```

参数

*prop_id*

[输入] 图形对象的字符串属性值。

*modifier=-1*

[输入] 限定符（索引）字符串特征。

返回值

字符串属性值。

验证处治的正确性再获取属性值

```
bool  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,       // 字符串属性标识符
   int                          modifier,      // 限定符
   string&                      value          // 链接到变量
   ) const
```

参数

*prop_id*

[输入] 图形对象的字符串属性值。

*modifier*

[输入] 限定符（索引）字符串特征。

*value*

[输出] 容纳属性字符串值的变量引用。

返回值

true - 如果成功，false - 如果您未能获取一个字符串特征值。

例如：

```
//--- 例程 CChartObject::GetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
   CChartObject object;
   string       value;
   //--- 通过简便方法获取图表名
   printf("对象名是 '%s'",object.GetString(OBJPROP_NAME));
   //--- 通过经典方法获取图表名
   if(!object.GetString(OBJPROP_NAME,0,value))
     {
      printf("获取字符串属性出错 %d",GetLastError());
      return;
     }
   else
      printf("对象名是 '%s'",value);
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- 通过简便方法获取水平线描述
      printf("水平线 %d 描述是 '%s'",i,object.GetString(OBJPROP_LEVELTEXT,i));
      //--- 通过经典方法获取水平线描述
      if(!object.GetString(OBJPROP_LEVELTEXT,i,value))
        {
         printf("获取字符串属性出错 %d",GetLastError());
         return;
        }
      else
         printf("水平线 %d 描述是 '%s'",i,value);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# SetString

可供简便地访问 MQL5 API ObjectSetString() 函数来获取绑定实例的图形类的字符串属性值。函数有两个调用版本：

无需限定符即可修改一个属性值

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,      // 字符串属性标识符
   string                        value         // 值
   )
```

参数

*prop_id*

[输入] 图形对象的字符串属性值。

*value*

[输入] 字符串特征的新可变值。

根据限定符设置一个属性值

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,       // 字符串属性标识符
   int                           modifier,      // 限定符
   string                        value          // 值
   )
```

参数

*prop_id*

[输入] 图形对象的字符串属性值。

*modifier*

[输入] 限定符（索引）字符串特征。

*value*

[输入] 字符串特征的新可变值。

返回值

true - 如果成功，false - 如果您未能改变字符串特征值。

例如：

```
//--- 例程 CChartObject::SetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 设置图表对象新名称
   if(!object.SetString(OBJPROP_NAME,"MyObject"))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       printf("设置字符串属性错误 %d",GetLastError());
       return;
      }
   for(int i=0;i<object.LevelsCount();i++)
      {
       //--- 设置水平线描述
       if(!object.SetString(OBJPROP_LEVELTEXT,i,"Level_"+IntegerToString(i)))
          {
           printf("设置字符串属性错误 %d",GetLastError());
           return;
          }
      }
  }
```

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象的参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CChartObject::Save
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CChartObject object=new CChartObject;
   //--- 设置对象参数
   //--- . . .
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true - 如果成功完成，false - 如果出错。

例如：

```
//--- 例程 CChartObject::Load
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CChartObject object;
   //--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- 文件加载出错
         printf("文件加载: 错误 %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- 使用对象
   //--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

获取图形对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（0x8888 对应 CChartObject）。

例如：

```
//--- 例程 CChartObject::Type
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- 获取对象类型
   int type=object.Type();
  }
```

© 2000-2025, MetaQuotes Ltd.

# 线条对象

"线条"图形对象组

这章节包含操作图形对象"线条"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectVLine | 图形对象"垂直线" |
| CChartObjectHLine | 图形对象"水平线" |
| CChartObjectTrend | 图形对象"趋势线" |
| CChartObjectTrendByAngle | 图形对象"角度趋势线" |
| CChartObjectCycles | 图形对象"圆形线" |

参阅

对象类型，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectVLine

CChartObjectVLine 类用于简便地访问 "垂直线" 图形对象属性。

## 描述

CChartObjectVLine 类可供访问 "垂直线" 对象属性。

## 声明

```
class CChartObjectVLine : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectVLine

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "垂直线" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"垂直线"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time           // 时间坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time*

[输入] 锚点的时间坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符 (OBJ_VLINE 对应 CChartObjectVLine)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectHLine

CChartObjectHLine 类用于简便地访问 "水平线" 图形对象属性。

## 描述

CChartObjectHLine 类可供访问 "水平线" 对象属性。

## 声明

```
class CChartObjectHLine : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectHLine

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "水平线" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"水平线"。

```
bool  Create(
   long    chart_id,      // 图表标识符
   string  name,          // 对象名称
   long    window,        // 图表窗口
   double  price          // 价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*price*

[输入] 锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_HLINE 对应 CChartObjectHLine）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectTrend

CChartObjectTrend 类用于简便地访问 "趋势线" 图形对象属性。

## 描述

CChartObjectTrend 类可供访问 "趋势线" 对象属性。

## 声明

```
class CChartObjectTrend : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend

直接分支

CChartObjectChannel, CChartObjectFibo, CChartObjectFiboChannel, CChartObjectFiboExpansion, CChartObjectGannFan, CChartObjectGannGrid, CChartObjectPitchfork, CChartObjectRegression, CChartObjectStdDevChannel, CChartObjectTrendByAngle

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "趋势线" |
| 属性 | |
| RayLeft | 获取/设置 "左侧射线" 属性 |
| RayRight | 获取/设置 "右侧射线" 属性 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"趋势线"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## RayLeft (Get 方法)

获取 "左侧射线" 属性值。

```
bool  RayLeft() const
```

返回值

已分配到类实例中的 "左侧射线" 属性值。如果无分配对象，则返回 false。

## RayLeft (Set 方法)

设置 "左侧射线" 属性新值。

```
bool  RayLeft(
   bool  ray      // 标志
   )
```

参数

*ray*

[输入] 新的 "左侧射线" 属性值。

返回值

true 如果成功，false 如果标志未改变。

© 2000-2025, MetaQuotes Ltd.

## RayRight (Get方法)

获取"右侧射线"属性值。

```
bool  RayRight() const
```

返回值

已分配到类实例中的"右侧射线"属性值。如果无分配对象，则返回 false。

## RayRight (Set方法)

设置"右侧射线"属性新值。

```
bool  RayRight(
   bool  ray      // 标志
   )
```

参数

*ray*

[输入] 新的"右侧射线"属性值。

返回值

true 如果成功，false 如果标志未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符 (OBJ_TREND 对应 CChartObjectTrend)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectTrendByAngle

CChartObjectTrendByAngle 类用于简便地访问 "角度趋势线" 图形对象属性。

## 描述

CChartObjectTrendByAngle 类可供访问 "角度趋势线" 对象属性。

## 声明

```
class CChartObjectTrendByAngle : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

## 继承体系

CObject
CChartObject
CChartObjectTrend
CChartObjectTrendByAngle

直接分支

CChartObjectGannLine

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "角度趋势线" |
| 属性 | |
| Angle | 获取/设置 属性 "角度" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"角度趋势线"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   long       window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## Angle (Get方法)

获取"锚点"属性的值。

```
double  Angle() const
```

返回值

已分配到类实例中的"角度"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## Angle (Set方法)

设置新的"角度"属性值。

```
bool  Angle(
   double  angle      // 角度
   )
```

参数

*angle*

[输入] 新的"角度"属性值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符 (OBJ_TRENDBYANGLE 对应 CChartObjectTrendByAngle)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectCycles

CChartObjectCycles 类用于简便地访问 "圆形线" 图形对象属性。

## 描述

CChartObjectCycles 类可供访问 "圆形线" 对象属性。

## 声明

```
class CChartObjectCycles : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectCycles

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "圆形线" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"圆形线"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   long       window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_CYCLES 对应 CChartObjectCycles）。

© 2000-2025, MetaQuotes Ltd.

# 通道对象

"通道"图形对象组

这章节包含操作图形对象"通道"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectChannel | 图形对象"等距通道" |
| CChartObjectRegression | 图形对象"线性回归通道" |
| CChartObjectStdDevChannel | 图形对象"标准方差通道" |
| CChartObjectPitchfork | 图形对象"安德鲁草叉" |

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectChannel

CChartObjectChannel类用于简便地访问"等距通道"图形对象属性。

## 描述

CChartObjectChannel类可供访问"等距通道"对象属性。

## 声明

```
class CChartObjectChannel : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectChannel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象"等距通道" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"等距通道"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个锚点的时间坐标
   double     price1,        // 第一个锚点的价格坐标
   datetime   time2,         // 第二个锚点的时间坐标
   double     price2,        // 第二个锚点的价格坐标
   datetime   time3,         // 第三个锚点的时间坐标
   double     price3         // 第三个锚点的价格坐标
   )
```

参数

*chart_id*

[输入] 图表 ID (0 -当前图表)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_CHANNEL 对应 CChartObjectChannel）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectRegression

CChartObjectRegression 类用于简便地访问 "线性回归通道" 图形对象属性。

## 描述

CChartObjectRegression 类可供访问 "线性回归通道" 对象属性。

## 声明

```
class CChartObjectRegression : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectRegression

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "线性回归通道" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类 CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"线性回归通道"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   long       window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   datetime   time2          // 第二个时间坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*time2*

[输入] 第二个锚点的时间坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_REGRESSION 对应 CChartObjectRegression）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectStdDevChannel

CChartObjectStdDevChannel类用于简便地访问"标准方差通道"图形对象属性。

## 描述

CChartObjectStdDevChannel类可供访问"标准方差通道"对象属性。

## 声明

```
class CChartObjectStdDevChannel : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectStdDevChannel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象"标准方差通道" |
| 属性 | |
| Deviations | 获取/设置"方差"属性 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象 "标准方差通道"

```
bool  Create(
   long       chart_id,       // 图表标识符
   string     name,           // 对象名称
   int        window,         // 图表窗口
   datetime   time1,          // 第一个时间坐标
   datetime   time2,          // 第二个时间坐标
   double     deviation       // 方差
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*deviation*

[输入] "方差"属性的数值。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## Deviation (Get方法)

获取"方差"属性的数值。

```
double  Deviation() const
```

返回值

已分配到类实例中的"方差"属性的数值。如果无分配对象，则返回 EMPTY_VALUE。

## Deviation (Set方法)

设置"方差"属性的数值。

```
bool  Deviation(
   double  deviation      // 方差
   )
```

参数

*deviation*

[输入]"方差"属性的新数值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符 (OBJ_STDDEVCHANNEL 对应 CChartObjectStdDevChannel)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectPitchfork

CChartObjectPitchfork 类用于简便地访问 "安德鲁草叉" 图形对象属性。

## 描述

CChartObjectPitchfork 类可供访问 "安德鲁草叉" 对象属性。

## 声明

```
class CChartObjectPitchfork : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectPitchfork

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "安德鲁草叉" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类 CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"安德鲁草叉"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   long       window,        // 图表窗口
   datetime   time1,         // 第一个锚点的时间坐标
   double     price1,        // 第一个锚点的价格坐标
   datetime   time2,         // 第二个锚点的时间坐标
   double     price2,        // 第二个锚点的价格坐标
   datetime   time3,         // 第三个锚点的时间坐标
   double     price3         // 第三个锚点的价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符 (OBJ_PITCHFORK 对应 CChartObjectPitchfork)。

© 2000-2025, MetaQuotes Ltd.

# 江恩工具

"江恩工具"图形对象组

这章节包含操作图形对象"江恩工具"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectGannLine | 图形对象"江恩线" |
| CChartObjectGannFan | 图形对象"江恩扇形" |
| CChartObjectGannGrid | 图形对象"江恩网格" |

参阅

对象类型，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectGannLine

CChartObjectGannLine 类用于简便地访问 "江恩线" 图形对象属性。

## 描述

CChartObjectGannLine 类可供访问 "江恩线" 对象属性。

## 声明

```
class CChartObjectGannLine : public CChartObjectTrendByAngle
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectTrendByAngle
        CChartObjectGannLine

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "江恩线" |
| 属性 | |
| PipsPerBar | 获取/设置 "缩放" 属性。 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

方法继承自类CChartObjectTrendByAngle

Angle, Angle, Create

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"江恩线"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     ppb            // 每根柱线的点数
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*ppb*

[输入] 每根柱线的点数。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## PipsPerBar (Get方法)

获取"每根柱线点数"属性的值。

```
double  PipsPerBar() const
```

返回值

已分配到类实例中的对象"每根柱线点数"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## PipsPerBar (Set方法)

设置"每根柱线点数"属性的值。

```
bool  PipsPerBar(
   double  ppb       // 每根柱线点数
   )
```

参数

*ppb*

[输入]"每根柱线点数"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_GANNLINE 对应 CChartObjectGannLine)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectGannFan

CChartObjectGannFan 类用于简便地访问 "江恩扇形" 图形对象属性。

## 描述

CChartObjectGannFan 类可供访问 "江恩扇形" 对象属性。

## 声明

```
class CChartObjectGannFan : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectGannFan

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "江恩扇形" |
| 属性 | |
| PipsPerBar | 获取/设置 "每根柱线点数" 属性 |
| Downtrend | 获取/设置 "下跌趋势" 属性。 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"江恩扇形"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time1,         // 第一个时间坐标
   double    price1,        // 第一个价格坐标
   datetime  time2,         // 第二个时间坐标
   double    ppb            // 每根柱线的点数
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*ppb*

[输入] 每根柱线的点数。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## PipsPerBar (Get方法)

获取"每根柱线点数"属性的值。

```
double  PipsPerBar() const
```

返回值

已分配到类实例中的对象"每根柱线点数"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## PipsPerBar (Set方法)

设置"每根柱线点数"属性的值。

```
bool  PipsPerBar(
   double  ppb       // 每根柱线点数
   )
```

参数

*ppb*

[输入]"每根柱线点数"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Downtrend (Get方法)

获取"下跌趋势"属性的值。

```
bool  Downtrend() const
```

返回值

已分配到类实例中的对象"下跌趋势"属性值。如果无分配对象，则返回 false。

## Downtrend (Set方法)

设置"下跌趋势"属性的新值。

```
bool  Downtrend(
   bool  downtrend      // 标志值
   )
```

参数

*downtrend*

[输入]"下跌趋势"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（OBJ_GANNFAN 对应 CChartObjectGannFan）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectGannGrid

CChartObjectGannGrid 类用于简便地访问 "江恩网格" 图形对象属性。

## 描述

CChartObjectGannGrid 类可供访问 "江恩网格" 对象属性。

## 声明

```
class CChartObjectGannGrid : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectGannGrid

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "江恩网格" |
| 属性 | |
| PipsPerBar | 获取/设置 "每根柱线点数" 属性 |
| Downtrend | 获取/设置 "下跌趋势" 属性。 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"江恩网格"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time1,         // 第一个时间坐标
   double    price1,        // 第一个价格坐标
   datetime  time2,         // 第二个时间坐标
   double    ppb            // 每根柱线的点数
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*ppb*

[输入] 每根柱线的点数。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## PipsPerBar (Get方法)

获取"每根柱线点数"属性的值。

```
double  PipsPerBar() const
```

返回值

已分配到类实例中的对象"每根柱线点数"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## PipsPerBar (Set方法)

设置"每根柱线点数"属性的值。

```
bool  PipsPerBar(
   double  ppb       // 每根柱线点数
   )
```

参数

*ppb*

[输入]"每根柱线点数"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Downtrend (Get方法)

获取"下跌趋势"属性的值。

```
bool  Downtrend() const
```

返回值

已分配到类实例中的对象"下跌趋势"属性值。如果无分配对象，则返回 false。

## Downtrend (Set方法)

设置"下跌趋势"属性的新值。

```
bool  Downtrend(
   bool  downtrend      // 标志值
   )
```

参数

*downtrend*

[输入]"下跌趋势"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（OBJ_GANNGRID 对应 CChartObjectGannGrid）。

© 2000-2025, MetaQuotes Ltd.

# 菲波纳奇工具

"菲波纳奇工具"图形对象组

这章节包含操作图形对象"菲波纳奇工具"类的技术细节，以及 MQL5 标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectFibo | 图形对象"菲波纳奇回撤" |
| CChartObjectFiboTimes | 图形对象"菲波纳奇时区" |
| CChartObjectFiboFan | 图形对象"菲波纳奇扇形" |
| CChartObjectFiboArc | 图形对象"菲波纳奇弧形" |
| CChartObjectFiboChannel | 图形对象"菲波纳奇通道" |
| CChartObjectFiboExpansion | 图形对象"菲波纳奇扩展" |

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFibo

CChartObjectFibo 类用于简便地访问 "菲波纳奇回撤" 图形对象属性。

## 描述

CChartObjectFibo 类可供访问 "菲波纳奇回撤" 对象属性。

## 声明

```
class CChartObjectFibo : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFibo

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "菲波纳奇回撤" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类 CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇回撤"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time1,         // 第一个时间坐标
   double    price1,        // 第一个价格坐标
   datetime  time2,         // 第二个时间坐标
   double    price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_FIBO 对应 CChartObjectFibo)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFiboTimes

CChartObjectFiboTimes 类用于简便地访问 "菲波纳奇时区" 图形对象属性。

## 描述

CChartObjectFiboTimes 类可供访问 "菲波纳奇时区" 对象属性。

## 声明

```
class CChartObjectFiboTimes : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectFiboTimes

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "菲波纳奇时区" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇时区"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（OBJ_FIBOTIMES 对应 CChartObjectFiboTimes）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFiboFan

CChartObjectFiboFan 类用于简便地访问 "菲波纳奇扇形" 图形对象属性。

## 描述

CChartObjectFiboFan 类可供访问 "菲波纳奇扇形" 对象属性。

## 声明

```
class CChartObjectFiboFan : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectFiboFan

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "菲波纳奇扇形" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇扇形"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（OBJ_FIBOFAN 对应 CChartObjectFiboFan）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFiboArc

CChartObjectFiboArc 类用于简便地访问 "菲波纳奇弧形" 图形对象属性。

## 描述

CChartObjectFiboArc 类可供访问 "菲波纳奇弧形" 对象属性。

## 声明

```
class CChartObjectFiboArc : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectFiboArc

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "菲波纳奇弧形" |
| 属性 | |
| Scale | 获取/设置 "缩放" 属性。 |
| Ellipse | 获取/设置 "椭圆形" 属性。 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

© 2000-2025, MetaQuotes Ltd.

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇弧形"

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   double     scale          // 缩放
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*scale*

[输入] 缩放。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## Scale (Get方法)

获取 "Scale" 属性的值。

```
double  Scale() const
```

返回值

已分配到类实例中的对象 "Scale" 属性值。如果无分配对象，则返回 EMPTY_VALUE。

## Scale (Set方法)

设置 "Scale" 属性的新值。

```
bool  Scale(
   double  scale      // 缩放
   )
```

参数

*scale*

[输入] "Scale" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Ellipse (Get方法)

获取"椭圆形"属性的值。

```
bool  Ellipse() const
```

返回值

已分配到类实例中的对象"椭圆形"属性值。如果无分配对象，则返回 false。

## Ellipse (Set方法)

设置"椭圆形"属性的新值。

```
bool  Ellipse(
   bool  ellipse      // 标志值
   )
```

参数

*ellipse*

[输入]"Scale"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_FIBOARC 对应 CChartObjectFiboArc)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFiboChannel

CChartObjectFiboChannel类用于简便地访问"菲波纳奇通道"图形对象属性。

## 描述

CChartObjectFiboChannel类可供访问"菲波纳奇通道"对象属性。

## 声明

```
class CChartObjectFiboChannel : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFiboChannel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象"菲波纳奇通道" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇通道"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   datetime   time3,         // 第三个时间坐标
   double     price3         // 第三个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_FIBOCHANNEL 对应 CChartObjectFiboChannel)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectFiboExpansion

CChartObjectFiboExpansion 类用于简便地访问 "菲波纳奇扩展" 图形对象属性。

## 描述

CChartObjectFiboExpansion 类可供访问 "菲波纳奇扩展" 对象属性。

## 声明

```
class CChartObjectFiboExpansion : public CChartObjectTrend
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFiboExpansion

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "菲波纳奇扩展" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类 CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"菲波纳奇扩展"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   datetime   time3,         // 第三个时间坐标
   double     price3         // 第三个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_EXPANSION 对应 CChartObjectFiboExpansion)。

© 2000-2025, MetaQuotes Ltd.

# 埃洛特工具

"埃洛特工具"图形对象组

这章节包含操作图形对象"埃洛特工具"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectElliottWave3 | 图形对象"调整浪" |
| CChartObjectElliottWave5 | 图形对象"推动浪" |

参阅

对象类型，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectElliottWave3

CChartObjectElliottWave3 类用于简便地访问 "调整浪" 图形对象属性。

## 描述

CChartObjectElliottWave3 类可供访问 "调整浪" 对象属性。

## 声明

```
class CChartObjectElliottWave3 : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectElliottWave3

直接分支

CChartObjectElliottWave5

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "调整浪" |
| 属性 | |
| Degree | 获取/设置 "度数" 属性。 |
| Lines | 获取/设置 "线条" 属性。 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"调整浪"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   datetime   time3,         // 第三个时间坐标
   double     price3         // 第三个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的时间坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## Degree (Get方法)

获取"度数"属性的值。

```
ENUM_ELLIOT_WAVE_DEGREE  Degree() const
```

返回值

已分配到类实例中的对象"度数"属性值。如果无分配对象，则返回 WRONG_VALUE。

## Degree (Set方法)

设置"度数"属性的新值。

```
bool  Degree(
   ENUM_ELLIOT_WAVE_DEGREE  degree      // 属性值
   )
```

参数

*degree*

[输入]"度数"属性新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Lines (Get 方法)

获取 "线条" 属性的值。

```
bool  Lines() const
```

返回值

已分配到类实例中的对象 "线条" 属性值。如果无分配对象，则返回 false。

## Lines (Set 方法)

设置 "线条" 属性的新值。

```
bool  Lines(
   bool  lines      // 标志值
   )
```

参数

*lines*

[输入] "线条" 属性的新值。

返回值

true 如果成功，false 如果标志未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_ELLIOTWAVE3 对应 CChartObjectElliottWave3)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectElliottWave5

CChartObjectElliottWave5 类用于简便地访问 "推动浪" 图形对象属性。

## 描述

CChartObjectElliottWave5 类可供访问 "推动浪" 对象属性。

## 声明

```
class CChartObjectElliottWave5 : public CChartObjectElliottWave3
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectElliottWave3
      CChartObjectElliottWave5

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "推动浪" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类 CChartObjectElliottWave3

Degree, Degree, Lines, Lines, Create, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"推动浪"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   datetime   time3,         // 第三个时间坐标
   double     price3,        // 第三个价格坐标
   datetime   time4,         // 第四个时间坐标
   double     price4,        // 第四个价格坐标
   datetime   time5          // 第五个时间坐标
   double     price5,        // 第五个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

*time4*

[输入] 第四个锚点的时间坐标。

© 2000-2025, MetaQuotes Ltd.

*price4*

[输入] 第四个锚点的价格坐标。

*time5*

[输入] 第五个锚点的时间坐标。

*price5*

[输入] 第五个锚点的价格坐标。

### 返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符（OBJ_ELLIOTWAVE5 对应 CChartObjectElliottWave5）。

© 2000-2025, MetaQuotes Ltd.

# 形状对象

"形状"图形对象组

这章节包含操作图形对象"形状"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectRectangle | 图形对象"长方形" |
| CChartObjectTriangle | 图形对象"三角形" |
| CChartObjectEllipse | 图形对象"椭圆形" |

参阅

对象类型，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectRectangle

CChartObjectRectangle 类用于简便地访问 "长方形" 图形对象属性。

## 描述

CChartObjectRectangle 类可供访问 "长方形" 对象属性。

## 声明

```
class CChartObjectRectangle : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectRectangle

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "长方形" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"长方形"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   long      window,        // 图表窗口
   datetime  time1,         // 第一个时间坐标
   double    price1,        // 第一个价格坐标
   datetime  time2,         // 第二个时间坐标
   double    price2         // 第二个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

返回值

true 如果成功，false 如果出错

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_RECTANGLE 对应 CChartObjectRectangle）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectTriangle

CChartObjectTriangle 类用于简便地访问 "三角形" 图形对象属性。

## 描述

CChartObjectTriangle 类可供访问 "三角形" 对象属性。

## 声明

```
class CChartObjectTriangle : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectTriangle

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "三角形" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"三角形"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   long       window,        // 图表窗口
   datetime   time1,         // 第一个时间坐标
   double     price1,        // 第一个价格坐标
   datetime   time2,         // 第二个时间坐标
   double     price2,        // 第二个价格坐标
   datetime   time3,         // 第三个时间坐标
   double     price3         // 第三个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_TRIANGLE 对应 CChartObjectTriangle）。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectEllipse

CChartObjectEllipse 类用于简便地访问 "椭圆形" 图形对象属性。

## 描述

CChartObjectEllipse 类可供访问 "椭圆形" 对象属性。

## 声明

```
class CChartObjectEllipse : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectEllipse

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "椭圆形" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

参阅

对象类型, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象 "椭圆形"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time1,         // 第一个时间坐标
   double    price1,        // 第一个价格坐标
   datetime  time2,         // 第二个时间坐标
   double    price2,        // 第二个价格坐标
   datetime  time3,         // 第三个时间坐标
   double    price3         // 第三个价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time1*

[输入] 第一个锚点的时间坐标。

*price1*

[输入] 第一个锚点的价格坐标。

*time2*

[输入] 第二个锚点的时间坐标。

*price2*

[输入] 第二个锚点的价格坐标。

*time3*

[输入] 第三个锚点的时间坐标。

*price3*

[输入] 第三个锚点的价格坐标。

返回值

true 如果成功，false 如果出错

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符。

```
int  Type() const
```

返回值

对象类型标识符（OBJ_ELLIPSE 对应 CChartObjectEllipse）。

© 2000-2025, MetaQuotes Ltd.

# 箭头对象

箭头图形对象组

这章节包含操作图形对象"箭头"类的技术细节，以及MQL5标准库中相关组件的描述。在本质上，箭头 - 是显示给用户的一些图标，而图标匹配某些代码。在图表上可显示两类"箭头"图形对象：

- "箭头"对象，可令您指定显示对象的图标代码。
- 对象组显示某类图标（且对应于某类固定代码）。

## 箭头类则显示任意代码图标。

| 类名称 | 箭头对象名称 |
| --- | --- |
| CChartObjectArrow | 箭头 |

## 固定代码图标的箭头类

| 类名称 | 箭头对象名称 |
| --- | --- |
| CChartObjectArrowCheck | 勾选 |
| CChartObjectArrowDown | 上箭头 |
| CChartObjectArrowUp | 下箭头 |
| CChartObjectArrowStop | 停止标志 |
| CChartObjectArrowThumbDown | 拇指竖起 |
| CChartObjectArrowThumbUp | 拇指向下 |
| CChartObjectArrowLeftPrice | 左侧价格标签 |
| CChartObjectArrowRightPrice | 右侧价格标签 |

参阅

对象类型，绑定方法，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectArrow

CChartObjectArrow 类用于简便地访问 "箭头" 图形对象属性。

## 描述

CChartObjectArrow 类为其子类提供访问 "箭头" 对象的常规属性。

## 声明

```
class CChartObjectArrow : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectArrow

直接分支

CChartObjectArrowCheck, CChartObjectArrowDown, CChartObjectArrowLeftPrice, CChartObjectArrowRightPrice, CChartObjectArrowStop, CChartObjectArrowThumbDown, CChartObjectArrowThumbUp, CChartObjectArrowUp

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "箭头" |
| 属性 | |
| ArrowCode | 获取/设置 "箭头代码" 属性 |
| Anchor | 获取/设置 "锚点" 属性 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 对象绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象 "箭头"。

```
bool  Create(
   long      chart_id,      // 图表 ID
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time,          // 时间
   double    price,         // 价格
   char      code           // 箭头代码
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前图表)。

*name*

[输入] 对象名称 (应是独有)。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time*

[输入] 时间坐标。

*price*

[输入]价格坐标。

*code*

[输入] "箭头"代码 (W ingdings)。

返回值

true -如果成功，否则 false。

例如：

```
//--- 例程 CChartObjectArrow::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- 箭头创建错误
      printf("箭头创建: 错误 %d!",GetLastError());
      //---
```

© 2000-2025, MetaQuotes Ltd.

```
      return;
     }
//--- 使用箭头
//--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

## ArrowCode (Get方法)

获取"箭头"符号的代码。

```
char  ArrowCode() const
```

返回值

已分配到类实例的"箭头"对象的符号代码。如果无分配对象，则返回0。

## ArrowCode (Set方法)

设置"箭头"符号的代码

```
bool  ArrowCode(
   char  code      // 代码值
   )
```

参数

*code*

[输入] 新的"箭头"代码值 (Wingdings)。

返回值

true如果成功，false如果代码不可改变。

例如：

```
//--- 例程 CChartObjectArrow::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   char              code=181;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,code))
     {
      //--- 箭头创建错误
      printf("箭头创建: 错误 %d!",GetLastError());
      //---
      return;
     }
//--- 使用可能改变代码的箭头
//--- . . .
//--- 获取箭头代码
   if(arrow.ArrowCode()!=code)
     {
      //--- 设置箭头代码
      arrow.ArrowCode(code);
```

© 2000-2025, MetaQuotes Ltd.

```
      }
//--- 使用箭头
//--- . . .
   }
```

© 2000-2025, MetaQuotes Ltd.

## Anchor (Get 方法)

获取 "箭头" 对象的锚点类型

```
ENUM_ARROW_ANCHOR  Anchor() const
```

返回值

已分配到类实例的 "箭头" 对象锚点类型。如果无分配对象，则返回 WRONG_VALUE。

## Anchor (Set 方法)

设置 "Arrow" 对象的锚点类型

```
bool  Anchor(
   ENUM_ARROW_ANCHOR  anchor      // 新锚点类型
   )
```

参数

*anchor*

[输入] 新的锚点类型

返回值

true 如果成功，false 如果锚点不可改变。

例如：

```
//--- 例程 CChartObject::Anchor
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- 箭头创建错误
      printf("箭头创建: 错误 %d!",GetLastError());
      //---
      return;
     }
//--- 获取箭头锚点
   if(arrow.Anchor()!=anchor)
     {
     //--- 设置箭头锚点
     arrow.Anchor(anchor);
     }
//--- 使用箭头
```

© 2000-2025, MetaQuotes Ltd.

```
//--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，否则 false。

例如：

```
//--- 例程 CChartObjectArrow::Save
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int                file_handle;
   CChartObjectArrow arrow;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- 箭头创建错误
      printf("箭头创建: 错误 %d!",GetLastError());
      //---
      return;
     }
//--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Save(file_handle))
        {
         //--- 文件保存错误
         printf("文件保存: 错误 %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 已由 FileOpen(...) 函数打开的文件句柄。

返回值

true 如果成功，否则 false。

例如：

```
//--- 例程 CChartObjectArrow::Load
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int                file_handle;
   CChartObjectArrow arrow;
//--- 打开文件
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Load(file_handle))
        {
         //--- 文件加载错误
         printf("文件加载: 错误 %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
//--- 使用箭头
//--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符

```
virtual int  Type() const
```

返回值

对象类型标识符（例如，OBJ_ARROW CChartObjectArrow）

例如：

```
//--- 例程 CChartObjectArrow::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   //--- 获取箭头类型
   int type=arrow.Type();
  }
```

© 2000-2025, MetaQuotes Ltd.

# 固定代码箭头

"固定代码箭头"类用于简便地访问以下图形对象的属性：

| 类名称 | 箭头对象名称 |
|---|---|
| CChartObjectArrowCheck | "勾选箭头" |
| CChartObjectArrowDown | "下箭头" |
| CChartObjectArrowUp | "上箭头" |
| CChartObjectArrowStop | "停止箭头" |
| CChartObjectArrowThumbDown | "好"("拇指竖起") |
| CChartObjectArrowThumbUp | "坏"("拇指向下") |
| CChartObjectArrowLeftPrice | "左侧价格"箭头 |
| CChartObjectArrowRightPrice | "右侧价格"箭头 |

## 描述

"固定代码箭头"类可供访问对象属性。

## 声明

```
class CChartObjectArrowCheck      : public CChartObjectArrow;
class CChartObjectArrowDown       : public CChartObjectArrow;
class CChartObjectArrowUp         : public CChartObjectArrow;
class CChartObjectArrowStop       : public CChartObjectArrow;
class CChartObjectArrowThumbDown  : public CChartObjectArrow;
class CChartObjectArrowThumbUp    : public CChartObjectArrow;
class CChartObjectArrowLeftPrice  : public CChartObjectArrow;
class CChartObjectArrowRightPrice : public CChartObjectArrow;
```

## 标称库文件

```
<ChartObjects\ChartObjectsArrows.mqh>
```

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建指定的图形对象 |
| 属性 | |
| ArrowCode | 修改代码方法的"存根" |
| 输入/输出 | |
| virtual Type | 标识的虚方法 |

© 2000-2025, MetaQuotes Ltd.

参阅

对象类型, 对象绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建 "固定代码箭头" 图形对象。

```
bool  Create(
   long      chart_id,      // 图表 ID
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time,          // 时间
   double    price          // 价格
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time*

[输入] 时间坐标。

*price*

[输入]价格坐标。

返回值

true 如果成功，false 如果出错。

例如：

```
//--- 例程 CChartObjectArrowCheck::Create
//--- 例程 CChartObjectArrowDown::Create
//--- 例程 CChartObjectArrowUp::Create
//--- 例程 CChartObjectArrowStop::Create
//--- 例程 CChartObjectArrowThumbDown::Create
//--- 例程 CChartObjectArrowThumbUp::Create
//--- 例程 CChartObjectArrowLeftPrice::Create
//--- 例程 CChartObjectArrowRightPrice::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- 例程, 用 CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
```

© 2000-2025, MetaQuotes Ltd.

```
      {
       //--- 箭头创建错误
       printf("箭头创建: 错误 %d!",GetLastError());
       //---
       return;
      }
//--- 使用箭头
//--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

# ArrowCode

禁止 "Arrow" 代码变更。

```
bool  ArrowCode(
   char  code      // 代码值
   )
```

参数

*code*

[输入] 任何值

返回值

一直为 false。

例如：

```
//--- 例程 CChartObjectArrowCheck::ArrowCode
//--- 例程 CChartObjectArrowDown::ArrowCode
//--- 例程 CChartObjectArrowUp::ArrowCode
//--- 例程 CChartObjectArrowStop::ArrowCode
//--- 例程 CChartObjectArrowThumbDown::ArrowCode
//--- 例程 CChartObjectArrowThumbUp::ArrowCode
//--- 例程 CChartObjectArrowLeftPrice::ArrowCode
//--- 例程 CChartObjectArrowRightPrice::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- 例程, 用 CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- 设置对象参数
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
     {
      //--- 箭头创建错误
      printf("箭头创建: 错误 %d!",GetLastError());
      //---
      return;
     }
//--- 设置箭头代码
   if(!arrow.ArrowCode(181))
     {
      //--- 这不是错误
      printf("箭头代码不可改变");
     }
//--- 使用箭头
//--- . . .
  }
```

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型的标识符

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_ARROW_CHECK 对应 CChartObjectArrowCheck, OBJ_ARROW_DOWN 对应 CChartObjectArrowDown, OBJ_ARROW_UP 对应 CChartObjectArrowUp, OBJ_ARROW_STOP 对应 CChartObjectArrowStop, OBJ_ARROW_THUMB_DOWN 对应 CChartObjectArrowThumbDown, OBJ_ARROW_THUMB_UP 对应 CChartObjectArrowThumbUp, OBJ_ARROW_LEFT_PRICE 对应 CChartObjectArrowLeftPrice, OBJ_ARROW_RIGHT_PRICE 对应 CChartObjectArrowRightPrice)。

例如：

```
//--- 例程 CChartObjectArrowCheck::Type
//--- 例程 CChartObjectArrowDown::Type
//--- 例程 CChartObjectArrowUp::Type
//--- 例程 CChartObjectArrowStop::Type
//--- 例程 CChartObjectArrowThumbDown::Type
//--- 例程 CChartObjectArrowThumbUp::Type
//--- 例程 CChartObjectArrowLeftPrice::Type
//--- 例程 CChartObjectArrowRightPrice::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- 例程, 用 CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- 获取箭头类型
   int type=arrow.Type();
  }
```

© 2000-2025, MetaQuotes Ltd.

# 控件对象

"控件"图形对象组

这章节包含操作图形对象"对象控件"类的技术细节，以及MQL5标准库中相关组件的描述。

| 类名称 | 对象 |
| --- | --- |
| CChartObjectText | 图形对象"文本" |
| CChartObjectLabel | 图形对象"文本标签" |
| CChartObjectEdit | 图形对象"编辑" |
| CChartObjectButton | 图形对象"按钮" |
| CChartObjectSubChart | 图形对象"图表" |
| CChartObjectBitmap | 图形对象"位图" |
| CChartObjectBmpLabel | 图形对象"位图标签" |
| CChartObjectRectLabel | 图形对象"长方形标签" |

参阅

对象类型，图形对象

© 2000-2025, MetaQuotes Ltd.

# CChartObjectText

CChartObjectText 类用于简便地访问 "文本" 图形对象属性。

## 描述

CChartObjectText 类可供访问 "文本" 对象属性。

## 声明

```
class CChartObjectText : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectText

直接分支

CChartObjectLabel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "文本"。 |
| 属性 | |
| Angle | 获取/设置 属性 "角度" |
| Font | 获取/设置 属性 "字体" |
| FontSize | 获取/设置 属性 "字号" |
| Anchor | 获取/设置 "锚点" 属性 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

派生类:

- CChartObjectLabel

参阅

对象类型, 对象属性, 对象绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"文本"。

```
bool  Create(
   long       chart_id,      // 图表标识符
   string     name,          // 对象名称
   int        window,        // 图表窗口
   datetime   time,          // 时间坐标
   double     price          // 价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time*

[输入] 锚点的时间坐标。

*price*

[输入] 锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## Angle (Get方法)

获取"锚点"属性的值。

```
double  Angle() const
```

返回值

已分配到类实例中的对象"锚点"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## Angle (Set方法)

设置"锚点"属性的新值。

```
bool  Angle(
   double  angle      // 新角度
   )
```

参数

*angle*

[输入]"角度"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Font (Get方法)

获取"字体"属性的值。

```
string  Font() const
```

返回值

已分配到类实例中的对象"字体"属性值。如果无分配对象，则返回""。

## Font (Set方法)

设置"字体"属性的新值。

```
bool  Font(
   string  font      // 新字体
   )
```

参数

*font*

[输入]"字体"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## FontSize (Get方法)

获取"字号"属性的值。

```
int  FontSize() const
```

返回值

已分配到类实例中的对象"字号"属性值。如果无分配对象，则返回 0。

## FontSize (Set方法)

设置"字号"属性的新值。

```
bool  FontSize(
   int  size      // 新字号
   )
```

参数

*size*

[输入]"字体"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Anchor (Get方法)

获取"锚点"属性的值。

```
ENUM_ANCHOR_POINT  Anchor() const
```

返回值

已分配到类实例中的对象"锚点"属性值。如果无分配对象，则返回 WRONG_VALUE。

## Anchor (Set方法)

设置"锚点"属性的新值。

```
bool  Anchor(
   ENUM_ANCHOR_POINT  anchor      // 新值
   )
```

参数

*anchor*

[输入]"锚点"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_TEXT 对应 CChartObjectText)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectLabel

CChartObjectLabel类用于简便地访问"标签"图形对象属性。

## 描述

CChartObjectLabel类可供访问"标签"对象属性。

## 声明

```
class CChartObjectLabel : public CChartObjectText
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

## 继承体系

CObject
CChartObject
CChartObjectText
CChartObjectLabel

直接分支

CChartObjectEdit, CChartObjectRectLabel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象"标签"。 |
| 属性 | |
| X_Distance | 获取/设置属性"X_距离" |
| Y_Distance | 获取/设置属性"Y_距离" |
| X_Size | 获取/设置属性"X_大小" |
| Y_Size | 获取/设置属性"Y_大小" |
| Corner | 获取/设置属性"边角" |
| Time | 时间坐标变化的"存根" |
| Price | 价格坐标变化的"存根" |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

参阅

对象类型, 对象属性, 图表角度, 绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"标签"。

```
bool  Create(
   long     chart_id,      // 图表标识符
   string   name,          // 对象名称
   int      window,        // 图表窗口
   int      X,             // X 坐标
   int      Y              // Y 坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*X*

[输入] X 坐标。

*Y*

[输入] Y 坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## X_Distance (Get方法)

获取"X_距离"属性的值。

```
int  X_Distance() const
```

返回值

已分配到类实例中的对象"X_距离"属性值。如果无分配对象，则返回0。

## X_Distance (Set方法)

设置"X_距离"属性的新值。

```
bool  X_Distance(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Distance (Get方法)

获取"Y_距离"属性的值。

```
int  Y_Distance() const
```

返回值

已分配到类实例中的对象"Y_距离"属性值。如果无分配对象，则返回0。

## Y_Distance (Set方法)

设置"Y_距离"属性的新值。

```
bool  Y_Distance(
   int  Y      // 新值
   )
```

参数

*Y*

[输入]"Y_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# X_Size

获取 "X_大小" 属性的值。

```
int  X_Size() const
```

返回值

已分配到类实例中的对象 "X_大小" 属性值。如果无分配对象，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# Y_Size

获取"Y_大小"属性的值。

```
int  Y_Size() const
```

返回值

已分配到类实例中的对象"Y_大小"属性值。如果无分配对象，则返回 0。

© 2000-2025, MetaQuotes Ltd.

## Corner (Get方法)

获取"边角"属性的值。

```
ENUM_BASE_CORNER  Corner() const
```

返回值

已分配到类实例中的对象"边角"属性值。如果无分配对象，则返回 WRONG_VALUE。

## Corner (Set方法)

设置"边角"属性的新值。

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // 新值
   )
```

参数

*corner*

[输入]"边角"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Time

禁止修改时间坐标。

```
bool  Time(
   datetime  time      // 任意值
   )
```

参数

*time*

[输入] 任意日期时间类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Price

禁止修改价格坐标。

```
bool  Price(
   double  price      // 任意值
   )
```

参数

*price*

[输入] 任意双精度类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_LABEL 对应 CChartObjectLabel)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectEdit

CChartObjectEdit 类用于简便地访问 "编辑" 图形对象属性。

## 描述

CChartObjectEdit 类可供访问 "编辑" 对象属性。

## 声明

```
class CChartObjectEdit : public CChartObjectLabel
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

## 继承体系

CObject
CChartObject
CChartObjectText
CChartObjectLabel
CChartObjectEdit

直接分支

CChartObjectButton

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建 "编辑" 图形对象 |
| 属性 | |
| TextAlign | 获取/设置 "文本对齐" 属性 |
| X_Size | 获取 "X 大小" 属性 |
| Y_Size | 获取 "Y 大小" 属性 |
| BackColor | 获取/设置 "背景颜色" 属性 |
| BorderColor | 获取/设置 "边框颜色" 属性 |
| ReadOnly | 获取/设置 "只读" 属性 |
| Angle | 获取/设置 "角度" 属性 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

方法继承自类CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

参阅

对象类型, 对象属性, 图表角度, 绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"编辑"。

```
bool  Create(
   long     chart_id,      // 图表标识符
   string   name,          // 对象名称
   int      window,        // 图表窗口
   int      X,             // X 坐标
   int      Y,             // Y 坐标
   int      sizeX,         // X 大小
   int      sizeY          // Y 大小
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*X*

[输入] X 坐标。

*Y*

[输入] Y 坐标。

*sizeX*

[输入] X 大小。

*sizeY*

[输入] Y 大小。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## TextAlign (Get方法)

获取"文本对齐"属性值 (文本对齐模式)。

```
ENUM_ALIGN_MODE  TextAlign() const
```

返回值

已分配到类实例中的对象"文本对齐"属性值。

## TextAlign (Set方法)

设置"文本对齐"属性值 (文本对齐模式)。

```
bool  TextAlign(
   ENUM_ALIGN_MODE  align      // 新值
   )
```

参数

*align*

[输入]"文本对齐"属性的新值。

返回值

true 如果成功, false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# X_Size

设置"X_大小"属性的新值。

```
bool  X_Size(
   int  size      // 新值
   )
```

参数

*size*

[输入]"X_大小"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Y_Size

设置"Y_大小"属性的新值。

```
bool  Y_Size(
   int  size      // 新值
   )
```

参数

*size*

[输入]"Y_大小"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## BackColor (Get方法)

获取"后背颜色"属性的值。

```
color  BackColor() const
```

返回值

已分配到类实例中的对象"后背颜色"属性值。如果无分配对象，则返回 CLR_NONE。

## BackColor (Set方法)

设置"后背颜色"属性的新值。

```
bool  BackColor(
   color  new_color      // 新背景颜色
   )
```

参数

*new_color*

[输入]"后背颜色"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## BorderColor (Get方法)

获取"边框颜色"属性的值。

```
color  BorderColor() const
```

返回值

已分配到类实例中的对象"边框颜色"属性值。如果无分配对象，则返回 CLR_NONE。

## BorderColor (Set方法)

设置"边框颜色"属性的新值。

```
bool  BorderColor(
   color  new_color      // 新边框颜色
   )
```

参数

*new_color*

[输入]"边框颜色"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ReadOnly (Get方法)

获取"只读"属性的值。

```
bool  ReadOnly() const
```

返回值

已分配到类实例中的对象"只读"属性值。如果无分配对象，则返回 false。

## ReadOnly (Set方法)

设置"只读"属性的新值。

```
bool  ReadOnly(
   const  bool  flag      // 新值
   )
```

参数

*flag*

[输入]"只读"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Angle

禁止修改"角度"属性。

```
bool  Angle(
   double  angle      // 任意值
   )
```

参数

*angle*

[输入] 任意双精度类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_EDIT 对应 CChartObjectEdit)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectButton

CChartObjectButton 类用于简便地访问 "按钮" 图形对象属性。

## 描述

CChartObjectButton 类可供访问 "按钮" 对象属性。

## 声明

```
class CChartObjectButton : public CChartObjectEdit
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

## 继承体系

CObject
CChartObject
CChartObjectText
CChartObjectLabel
CChartObjectEdit
CChartObjectButton

直接分支

CChartObjectPanel

## 类方法

| 创建 | |
|---|---|
| 创建 | 继承自类 CChartObjectEdit |
| 属性 | |
| 状态 | 获取/设置按钮状态（按下/释放） |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

方法继承自类CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

方法继承自类CChartObjectEdit

X_Size, Y_Size, BackColor, BackColor, BorderColor, BorderColor, ReadOnly, ReadOnly, TextAlign, TextAlign, Angle, Create

参阅

对象类型, 对象属性, 图表角度, 对象绑定方法, 图形对象

© 2000-2025, MetaQuotes Ltd.

## State (Get方法)

获取"状态"属性的值。

```
bool  State() const
```

返回值

已分配到类实例中的对象"状态"属性值。如果无分配对象，则返回 false。

## State (Set方法)

设置"状态"属性的新值。

```
bool  State(
   bool  state      // 新状态值
   )
```

参数

*X*

[输入]"状态"属性新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_BUTTON 对应 CChartObjectButton)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectSubChart

CChartObjectSubChart 类用于简便地访问 "图表" 图形对象属性。

## 描述

CChartObjectSubChart 类可供访问 "图表" 对象属性。

## 声明

```
class CChartObjectSubChart : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectSubChart.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectSubChart

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "图表"。 |
| 属性 | |
| X_Distance | 获取/设置 属性 "X_距离" |
| Y_Distance | 获取/设置 属性 "Y_距离" |
| Corner | 获取/设置 属性 "边角" |
| X_Size | 获取/设置 属性 "X_大小" |
| Y_Size | 获取/设置 属性 "Y_大小" |
| Symbol | 获取/设置 属性 "品名" |
| Period | 获取/设置 属性 "周期" |
| Scale | 获取/设置 "缩放" 属性。 |
| DateScale | 获取/设置 "显示日期缩放" 属性。 |
| PriceScale | 获取/设置 "显示价格缩放" 属性。 |
| Time | 时间坐标变化的 "存根" |
| Price | 价格坐标变化的 "存根" |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 对象属性, 图表角度, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"子图表"。

```
bool  Create(
   long     chart_id,      // 图表标识符
   string   name,          // 对象名称
   int      window,        // 图表窗口
   int      X,             // X 坐标
   int      Y,             // Y 坐标
   int      sizeX,         // X 大小
   int      sizeY          // Y 大小
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*X*

[输入] X 坐标。

*Y*

[输入] Y 坐标。

*sizeX*

[输入] X 大小。

*sizeY*

[输入] Y 大小。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## X_Distance (Get方法)

获取"X_距离"属性的值。

```
int  X_Distance() const
```

返回值

已分配到类实例中的对象"X_距离"属性值。如果无分配对象，则返回0。

## X_Distance (Set方法)

设置"X_距离"属性的新值。

```
bool  X_Distance(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Distance (Get方法)

获取"Y_距离"属性的值。

```
int  Y_Distance() const
```

返回值

已分配到类实例中的对象"Y_距离"属性值。如果无分配对象，则返回0。

## Y_Distance (Set方法)

设置"Y_距离"属性的新值。

```
bool  Y_Distance(
   int  Y      // 新值
   )
```

参数

*Y*

[输入]"Y_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Corner (Get方法)

获取"边角"属性的值。

```
ENUM_BASE_CORNER Corner() const
```

返回值

已分配到类实例中的对象"边角"属性值。如果无分配对象，则返回 WRONG_VALUE。

## Corner (Set方法)

设置"边角"属性的新值。

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // 新值
   )
```

参数

*corner*

[输入]"边角"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## X_Size (Get方法)

获取"X_大小"属性的值。

```
int  X_Size() const
```

返回值

已分配到类实例中的对象"X_大小"属性值。如果无分配对象，则返回 0。

## X_Size (Set方法)

设置"X_大小"属性的新值。

```
bool  X_Size(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_大小"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Size (Get方法)

获取 "Y_大小" 属性的值。

```
int  Y_Size() const
```

返回值

已分配到类实例中的对象 "Y_大小" 属性值。如果无分配对象，则返回 0。

## Y_Size (Set方法)

设置 "Y_大小" 属性的新值。

```
bool  Y_Size(
   int  Y      // 新值
   )
```

参数

*Y*

[输入] "Y_大小" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Symbol(Get方法)

获取"品名"属性的值。

```
string  Symbol() const
```

返回值

已分配到类实例中的对象"品名"属性值。如果无分配对象，则返回""。

## Symbol(Set方法)

设置"品名"属性的新值。

```
bool  Symbol(
   string  symbol      // 新品名
   )
```

参数

*symbol*

[输入]"品名"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Period (Get方法)

获取"周期"属性的值。

```
int  Period() const
```

返回值

已分配到类实例中的对象"周期"属性值。如果无分配对象，则返回0。

## Period (Set方法)

设置"周期"属性的新值。

```
bool  Period(
   int  period      // 新值
   )
```

参数

*period*

[输入] 属性"周期"的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Scale (Get方法)

获取"Scale"属性的值。

```
double  Scale() const
```

返回值

已分配到类实例中的对象"Scale"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## Scale (Set方法)

设置"Scale"属性的新值。

```
bool  Scale(
   double  scale      // 新缩放
   )
```

参数

*scale*

[输入]"Scale"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## DateScale (Get方法)

获取"日期缩放"属性的值。

```
bool  DateScale() const
```

返回值

已分配到类实例中的对象"DateScale"属性值。如果无分配对象，则返回 false。

## DateScale (Set方法)

设置"日期缩放"属性的新值。

```
bool  DateScale(
   bool  scale      // 新值
   )
```

参数

*scale*

[输入]"日期缩放"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## PriceScale (Get方法)

获取"价格缩放"属性的值。

```
bool  PriceScale() const
```

返回值

已分配到类实例中的对象"PriceScale"属性值。如果无分配对象，则返回 false。

## PriceScale (Set方法)

设置"价格缩放"属性的新值。

```
bool  PriceScale(
   bool  scale      // 新值
   )
```

参数

*scale*

[输入]"价格缩放"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Time

禁止修改时间坐标。

```
bool  Time(
   datetime  time      // 任意值
   )
```

参数

*time*

[输入] 任意日期时间类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

## Price

禁止修改价格坐标。

```
bool  Price(
   double  price      // 任意值
   )
```

参数

*price*

[输入] 任意双精度类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_CHART 对应 CChartObjectSubChart)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectBitmap

CChartObjectBitmap 类用于简便地访问 "位图" 图形对象属性。

## 描述

CChartObjectBitmap 类可供访问 "位图" 对象属性。

## 声明

```
class CChartObjectBitmap : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectBitmap

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象 "位图" |
| 属性 | |
| BmpFile | 获取/设置 "位图" 文件名 |
| X_Offset | 获取/设置 属性 "X_偏移" |
| Y_Offset | 获取/设置 属性 "Y_偏移" |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 对象属性, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"位图"。

```
bool  Create(
   long      chart_id,      // 图表标识符
   string    name,          // 对象名称
   int       window,        // 图表窗口
   datetime  time,          // 时间坐标
   double    price          // 价格坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*time*

[输入] 锚点的时间坐标。

*price*

[输入] 锚点的价格坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## BmpFile (Get方法)

获取"位图文件"属性的值。

```
string  BmpFile() const
```

返回值

已分配到类实例中的对象"位图文件"属性值。如果无分配对象，则返回 false。

## BmpFile (Set方法)

设置"位图文件"属性的新值。

```
bool  BmpFile(
   string  name      // 新文件名称
   )
```

参数

*name*

[输入]"位图文件"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## X_Offset (Get方法)

获取"X_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 X 坐标)。

```
int  X_Offset() const
```

返回值

已分配到类实例中的对象"X_偏移"属性值。如果无分配对象，则返回 0。

## X_Offset (Set方法)

设置"X_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 X 坐标)。此值为相对原始图像左上角的像素值。

```
bool  X_Offset(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_偏移"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Offset (Get方法)

获取"Y_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 Y 坐标)。

```
int  Y_Offset() const
```

返回值

已分配到类实例中的对象"Y_偏移"属性值。如果无分配对象，则返回 0。

## Y_Offset (Set方法)

设置"Y_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 Y 坐标)。此值为相对原始图像左上角的像素值。

```
bool  Y_Offset(
   int  Y      // 新值
   )
```

参数

Y

[输入]"Y_偏移"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_BITMAP 对应 CChartObjectBitmap)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectBmpLabel

CChartObjectBmpLabel类用于简便地访问"位图标签"图形对象属性。

## 描述

CChartObjectBmpLabel类可供访问"位图标签"对象属性。

## 声明

```
class CChartObjectBmpLabel : public CChartObject
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectBmpLabel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建图形对象"位图标签" |
| 属性 | |
| X_Distance | 获取/设置属性"X_距离" |
| Y_Distance | 获取/设置属性"Y_距离" |
| X_Offset | 获取/设置属性"X_偏移" |
| Y_Offset | 获取/设置属性"Y_偏移" |
| Corner | 获取/设置属性"边角" |
| X_Size | 获取/设置属性"X_大小" |
| Y_Size | 获取/设置属性"Y_大小" |
| BmpFileOn | 获取/设置"BmpFileOn"属性的按钮状态（按下） |
| BmpFileOff | 获取/设置"BmpFileOff"属性的按钮状态（释放） |
| 状态 | 获取/设置"按钮状态"属性（按下/释放） |
| Time | 时间坐标变化的"存根" |
| Price | 价格坐标变化的"存根" |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

参阅

对象类型, 对象属性, 图表角度, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建图形对象"位图标签"。

```
bool  Create(
   long     chart_id,      // 图表标识符
   string   name,          // 对象名称
   int      window,        // 图表窗口
   int      X,             // X 坐标
   int      Y              // Y 坐标
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*X*

[输入] X 坐标。

*Y*

[输入] Y 坐标。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

## X_Distance (Get方法)

获取"X_距离"属性的值。

```
int  X_Distance() const
```

返回值

已分配到类实例中的对象"X_距离"属性值。如果无分配对象，则返回0。

## X_Distance (Set方法)

设置"X_距离"属性的新值。

```
bool  X_Distance(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Distance (Get方法)

获取"Y_距离"属性的值。

```
int  Y_Distance() const
```

返回值

已分配到类实例中的对象"Y_距离"属性值。如果无分配对象，则返回0。

## Y_Distance (Set方法)

设置"Y_距离"属性的新值。

```
bool  Y_Distance(
   int  Y      // 新值
   )
```

参数

*Y*

[输入]"Y_距离"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## X_Offset (Get方法)

获取"X_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 X 坐标)。

```
int  X_Offset() const
```

返回值

已分配到类实例中的对象"X_偏移"属性值。如果无分配对象，则返回 0。

## X_Offset (Set方法)

设置"X_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 X 坐标)。此值为相对原始图像左上角的像素值。

```
bool  X_Offset(
   int  X      // 新值
   )
```

参数

*X*

[输入]"X_偏移"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Y_Offset (Get方法)

获取"Y_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 Y 坐标)。

```
int  Y_Offset() const
```

返回值

已分配到类实例中的对象"Y_偏移"属性值。如果无分配对象，则返回 0。

## Y_Offset (Set方法)

设置"Y_偏移"属性的值 (在图形对象中的长方形可视区域的左上角 Y 坐标)。此值为相对原始图像左上角的像素值。

```
bool  Y_Offset(
   int  Y      // 新值
   )
```

参数

Y

[输入]"Y_偏移"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Corner (Get方法)

获取"边角"属性的值。

```
ENUM_BASE_CORNER Corner() const
```

返回值

已分配到类实例中的对象"边角"属性值。如果无分配对象，则返回 WRONG_VALUE。

## Corner (Set方法)

设置"边角"属性的新值。

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // 新值
   )
```

参数

*corner*

[输入]"边角"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# X_Size

获取 "X_大小" 属性的值。

```
int  X_Size() const
```

返回值

已分配到类实例中的对象 "X_大小" 属性值。如果无分配对象，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# Y_Size

获取"Y_大小"属性的值。

```
int  Y_Size() const
```

返回值

已分配到类实例中的对象"Y_大小"属性值。如果无分配对象，则返回 0。

© 2000-2025, MetaQuotes Ltd.

## BmpFileOn (Get方法)

获取"BmpFileOn"属性的值。

```
string  BmpFileOn() const
```

返回值

已分配到类实例中的对象"BmpFileOn"属性值。如果无分配对象，则返回""。

## BmpFileOn (Set方法)

设置"BmpFileOn"属性的新值。

```
bool  BmpFileOn(
   string  name      // 文件名称
   )
```

参数

*name*

[输入]"BmpFileOn"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## BmpFileOff (Get方法)

获取"BmpFileOff"属性的值。

```
string  BmpFileOff() const
```

返回值

已分配到类实例中的对象"BmpFileOff"属性值。如果无分配对象，则返回""。

## BmpFileOff (Set方法)

设置"BmpFileOff"属性的新值。

```
bool  BmpFileOff(
   string  name      // 文件名称
   )
```

参数

*name*

[输入]"BmpFileOff"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## State (Get方法)

获取"状态"属性的值。

```
bool  State() const
```

返回值

已分配到类实例中的对象"状态"属性值。如果无分配对象，则返回 false。

## State (Set方法)

设置"状态"属性的新值。

```
bool  State(
   bool  state      // 新状态值
   )
```

参数

*state*

[输入]"状态"属性新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Time

禁止修改时间坐标。

```
bool  Time(
   datetime  time      // 任意值
   )
```

参数

*time*

[输入] 任意日期时间类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Price

禁止修改价格坐标。

```
bool  Price(
   double  price      // 任意值
   )
```

参数

*price*

[输入] 任意双精度类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_BITMAP_LABEL 对应 CChartObjectBmpLabel)。

© 2000-2025, MetaQuotes Ltd.

# CChartObjectRectLabel

CChartObjectRectLabel类用于简便地访问"长方形标签"图形对象属性。

## 描述

CChartObjectRectLabel类可供访问"长方形标签"对象属性。

## 声明

```
class CChartObjectRectLabel : public CChartObjectLabel
```

## 标称库文件

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

## 继承体系

CObject
  CChartObject
    CChartObjectText
      CChartObjectLabel
        CChartObjectRectLabel

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建"长方形标签"图形对象 |
| 属性 | |
| X_Size | 设置水平线大小 |
| Y_Size | 设置垂直线大小 |
| BackColor | 获取/设置背景颜色 |
| Angle | 方法存根 |
| BorderType | 获取/设置边框类型 |
| 输入/输出 | |
| virtual Save | 写文件的虚方法 |
| virtual Load | 从文件读取的虚方法 |
| virtual Type | 标识的虚方法 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CChartObject

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

方法继承自类CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

方法继承自类CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

参阅

对象类型, 对象属性, 图形对象

© 2000-2025, MetaQuotes Ltd.

# 创建

创建 "图表对象长方形标签" 图形对象。

```
bool  Create(
   long     chart_id,      // 图表 ID
   string   name,          // 对象名称
   int      window,        // 图表窗口
   int      X,             // X 坐标
   int      Y,             // Y 坐标
   int      sizeX,         // 水平线大小
   int      sizeY          // 垂直线大小
   )
```

参数

*chart_id*

[输入] 图表标识符 (0 -当前窗口)。

*name*

[输入] 即将创建的对象的独有名称。

*window*

[输入] 图表窗口号码 (0 -基准窗口)。

*X*

[输入] X 坐标。

*Y*

[输入] Y 坐标。

*sizeX*

[输入] 水平线大小。

*sizeY*

[输入] 垂直线大小。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# X_Size

设置 "X_大小" 属性的值。

```
bool  X_Size(
   int  size      // 水平线大小
   )
```

参数

*size*

[输入] 水平线大小。

返回值

true 如果成功，false 如果属性未改变。

注释

为获取 "X_大小" 和 "Y_大小" 属性的值，使用 X_Size 和 Y_Size 方法，方法均位于父类 CChartObjectLabel 之中。

© 2000-2025, MetaQuotes Ltd.

# Y_Size

设置 "Y_大小" 属性的值。

```
bool  Y_Size(
   int  size      // 垂直线大小
   )
```

参数

*size*

[输入] 新的垂直线大小。

返回值

true 如果成功，false 如果属性未改变。

注释

为获取 "X_大小" 和 "Y_大小" 属性的值，使用 X_Size 和 Y_Size 方法，方法均位于父类 CChartObjectLabel 之中。

© 2000-2025, MetaQuotes Ltd.

## BackColor

获取背景颜色。

```
color  BackColor() const
```

返回值

已挂载到类实例中的对象背景颜色。如果无分配对象，则返回 0。

## BackColor

设置背景颜色。

```
bool  BackColor(
   color  new_color      // 新颜色
   )
```

参数

*new_color*

[输入] 新背景颜色。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Angle

方法存根。

```
bool  Angle(
   double  angle      // 任意值
   )
```

参数

*angle*

[输入] 任意 双精度 类型值。

返回值

一直为 false。

© 2000-2025, MetaQuotes Ltd.

## BorderType

获取边框类型。

```
int  BorderType() const
```

返回值

已挂载到类实例中的对象边框类型。如果无分配对象，则返回 0。

## BorderType

设置边框类型。

```
bool  BorderType(
   int  type      // 边框类型
   )
```

参数

*type*

[输入] 新的边框类型

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (OBJ_RECTANGLE_LABEL 对应 CChartObjectRectangleLabel)。

© 2000-2025, MetaQuotes Ltd.

# 自定义图形

这部分提供处理自定义图形的工具。

它们的使用可以大大促进绘制自定义图表，图示和数据可视化。

这里有单独的创建图形对象和基本物的类，用于绘制各种类型的饼图和曲线。实现显示对象的多种选项：改变线条风格和颜色，填充，使用图表数据序列等等。

| 类 | 描述 |
| --- | --- |
| CCanvas | 简化创建自定义图像的类 |
| CChartCanvas | 实施意在绘制图表及其元素的类的基类 |
| CHistogramChart | 绘制直方图的类 |
| CLineChart | 绘制曲线的类 |
| CPieChart | 绘制饼图的类 |

© 2000-2025, MetaQuotes Ltd.

# CCanvas

CCanvas是一种用于简化创建自定义图像的类。

## 描述

CCanvas提供了图形源的创建（绑定或未绑定图表对象）以及绘制图形基本物件。

## 声明

```
class CCanvas
```

## 主题

```
#include <Canvas\Canvas.mqh>
```

## 继承体系

CCanvas

直接分支

CChartCanvas, CFlameCanvas

## 按组划分类函数

| 创建 | |
|---|---|
| Attach | 将OBJ_BITMAP_LABEL对象附加到CCanvas类的实例 |
| Create | 创建一个未绑定图表对象的图形源 |
| CreateBitmap | 创建一个绑定图表对象的图形源 |
| CreateBitmapLabel | 创建一个绑定图表对象的图形源 |
| Destroy | 破坏图形源 |
| 属性 | |
| ChartObjectName | 获得绑定图表对象的名称 |
| ResourceName | 获得图形源的名称 |
| Width | 获得图形源的宽度 |
| Height | 获得图形源的高度 |
| LineStyleSet | 设置线条样式 |
| 更新屏幕上的对象 | |
| Update | 显示屏幕上的变化 |
| Resize | 调整图形源的大小 |
| 擦除/填充颜色 | |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| Erase | 擦除或填充指定颜色 |
| 数据访问 | |
| PixelGet | 获得指定坐标点的颜色 |
| PixelSet | 设置指定坐标点的颜色 |
| 绘制基本物件 | |
| LineVertical | 绘制垂直线 |
| LineHorizontal | 绘制水平线 |
| Line | 绘制手绘线 |
| Polyline | 绘制多段线 |
| Polygon | 绘制多边形 |
| EllipseWu | 绘制一个矩形 |
| Circle | 绘制一个圆形 |
| Triangle | 绘制一个三角形 |
| Ellipse | 绘制一个椭圆形 |
| Arc | 绘制一个椭圆弧线 |
| Pie | 绘制一个椭圆扇形 |
| 绘制填充好的基本物件 | |
| FillRectangle | 绘制一个填充好的矩形 |
| FillCircle | 绘制一个填充好的圆形 |
| FillTriangle | 绘制一个填充好的三角形 |
| FillPolygon | 绘制一个填充多边形 |
| FillEllipse | 绘制一个填充椭圆形 |
| Fill | 填充一个区域 |
| 柔化绘制基本物件 | |
| PixelSetAA | 绘制一个像素 |
| LineAA | 绘制一条线 |
| PolylineAA | 绘制多段线 |
| PolygonAA | 绘制多边形 |
| TriangleAA | 绘制一个三角形 |
| CircleAA | 绘制一个圆形 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| EllipseAA | 绘制一个椭圆形 |
| LineWu | 绘制一条线 |
| PolylineWu | 绘制多段线 |
| PolygonWu | 绘制多边形 |
| TriangleWu | 绘制一个三角形 |
| CircleWu | 绘制一个圆形 |
| EllipseWu | 绘制一个椭圆形 |
| LineThick | 通过抗锯齿算法绘制一段具有指定宽度的手绘线。 |
| LineThickVertical | 通过抗锯齿算法绘制一段具有指定宽度的手绘垂直线。 |
| LineThickHorizontal | 通过抗锯齿算法绘制一段具有指定宽度的手绘水平线 |
| PolygonSmooth | 通过两种抗锯齿算法绘制一个指定宽度的多边形 |
| PolygonThick | 通过抗锯齿算法绘制一个指定宽度的多边形 |
| PolylineSmooth | 通过两种抗锯齿算法绘制一个指定宽度的多段线 |
| PolylineThick | 通过抗锯齿算法绘制一个指定宽度的多段线 |
| 文本 | |
| FontSet | 设置字体参数 |
| FontNameSet | 设置字体名称 |
| FontSizeSet | 设置字体大小 |
| FontFlagsSet | 设置字体标识 |
| FontAngleSet | 设置字体倾斜角度 |
| FontGet | 获得字体参数 |
| FontNameGet | 获得字体名称 |
| FontSizeGet | 获得字体大小 |
| FontFlagsGet | 获得字体标识 |
| FontAngleGet | 获得字体倾斜角度 |
| TextOut | 展示文本 |
| TextWidth | 获得文本宽度 |
| TextHeight | 获得文本高度 |
| TextSize | 获得文本大小 |
| 透明度 | |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| TransparentLevelSet | 设置透明度水平 |
| 输入/输出 | |
| LoadFromFile | 读取BMP格式的图像 |

© 2000-2025, MetaQuotes Ltd.

# Attach

从OBJ_BITMAP_LABEL对象获得图形资源并将其附加到CCanvas类的实例。

```
bool  Attach(
   const long          chart_id,                          // 图表标识符
   const string        objname,                           // 对象名称
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // 颜色处理方式
  );
```

为OBJ_BITMAP_LABEL对象创建一个图形资源并将其附加到CCanvas类的实例。

```
bool  Attach(
   const long          chart_id,                          // 图表标识符
   const string        objname,                           // 对象名称
   const int           width,                             // 以像素表示的图像宽度
   const int           height,                            // 以像素表示的图像高度
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // 颜色处理方式
  );
```

参数

*chart_id*

[out] 图表标识符。

*objname*

[in] 图形对象的名称。

*width*

[in] 资源中的图像宽度。

*height*

[in] 资源中的图像高度。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

[in] 阿尔法通道处理方法。默认忽略阿尔法通道。

返回值

成功情况下，返回true，如果附加对象失败，则返回false。

© 2000-2025, MetaQuotes Ltd.

# Arc

在(x1,y1)和(x2,y2)位置的矩形上绘制椭圆形弧线。直线从椭圆形中心横切弧形边界，延伸至(x3,y3)和(x4,y4)两个坐标点。

```
void  Arc(
   int          x1,       // 矩形左上角的 X 坐标
   int          y1,       // 矩形左上角的 Y 坐标
   int          x2,       // 矩形右下角的 X 坐标
   int          y2,       // 矩形右下角的 Y 坐标
   int          x3,       // 找到弧形边界的第一个点的 X 坐标
   int          y3,       // 找到弧形边界的第一个点的 Y 坐标
   int          x4,       // 找到弧形边界的第二个点的 X 坐标
   int          y4,       // 找到弧形边界的第二个点的 Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[in] 形成矩形的左上角的 X 坐标。

*y1*

[in] 形成矩形的左上角的 Y 坐标。

*x2*

[in] 形成矩形的右下角的 X 坐标。

*y2*

[in] 形成矩形的右下角的 Y 坐标。

*x3*

[in] 直线从矩形中心点开始绘制到第一个点的 X 坐标，以获得弧形边界。

*y3*

[in] 直线从矩形中心点开始绘制到第一个点的 Y 坐标，以获得弧形边界。

*x4*

[in] 直线从矩形中心点开始绘制到第二个点的 X 坐标，以获得弧形边界。

*y4*

[in] 直线从矩形中心点开始绘制到第二个点的 Y 坐标，以获得弧形边界。

*clr*

[in] ARGB 格式的颜色模式。使用ColorToARGB() 函数，将颜色转换为ARGB格式。

在矩形上，绘制中心点在(x,y)，半径为rx和ry的椭圆形弧线。使用通过fi3和fi4的夹角形成的射线从椭圆形中心横切弧形边界。

```
void  Arc(
   int          x,        // 椭圆形中心的 X 坐标
```

© 2000-2025, MetaQuotes Ltd.

```
   int          y,        // 椭圆形中心的 Y 坐标
   int          rx,       // X 轴上的椭圆形半径
   int          ry,       // Y 轴上的椭圆形半径
   int          fi3,      // 从椭圆形中心发出的射线角，定义弧形的第一个边界
   int          fi4,      // 从椭圆形中心发出的射线角，定义弧形的第二个边界
   const uint   clr       // 颜色
   );
```

在矩形上，绘制中心点在(x,y)，半径为rx和ry的椭圆形弧线，并返回弧形边界的坐标。使用通过fi3和fi4的夹角形成的射线从椭圆形中心横切弧形边界。

```
void  Arc(
   int          x,        // 椭圆形中心的 X 坐标
   int          y,        // 椭圆形中心的 Y 坐标
   int          rx,       // X 轴上的椭圆形半径
   int          ry,       // Y 轴上的椭圆形半径
   int          fi3,      // 从椭圆形中心发出的射线角，定义弧形的第一个边界
   int          fi4,      // 从椭圆形中心发出的射线角，定义弧形的第二个边界
   int&         x3,       // 第一个弧形边界的 X 坐标
   int&         y3,       // 第一个弧形边界的 Y 坐标
   int&         x4,       // 第二个弧形边界的 X 坐标
   int&         y4,       // 第二个弧形边界的 Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x*

[in] 椭圆形中心的 X 坐标。

*y*

[in] 椭圆形中心的 Y 坐标。

*rx*

[in] 像素表示的X 轴上的椭圆形半径。

*ry*

[in] 像素表示的X 轴上的椭圆形半径。

*fi3*

[in] 定义弧形第一个边界的弧度角。

*fi4*

[in] 定义弧形第二个边界的弧度角。

*x3*

[out] 获得第一个弧形边界 X 坐标的变量。

*y3*

[out] 获得第一个弧形边界 Y 坐标的变量。

*x4*

© 2000-2025, MetaQuotes Ltd.

[out] 获得第二个弧形边界 X 坐标的变量。

*y4*

[out] 获得第二个弧形边界 Y 坐标的变量。

*clr*

[in] ARGB 格式的颜色模式。使用ColorToARGB() 函数，将颜色转换为ARGB格式。

调用类函数的示例：

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int      Width=600;
   int      Height=400;
//--- 创建 canvas
   if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
     {
      Print("Error creating canvas: ",GetLastError());
     }
//--- 清除 canvas
   canvas.Erase(clrWhite);
//--- 绘制矩形
   canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- 绘制第一个弧形
   canvas.Arc(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrRed));
   int x1,y1,x2,y2;
//--- 绘制第二个弧形
   canvas.Arc(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,x1,y1,x2,y2,ColorToARGB(cl
//--- 打印弧形坐标
   PrintFormat("First point of arc at (%G,%G), second point of arc at (%G,%G)",x1,y1,x
   canvas.CircleAA(x1,y1,3, ColorToARGB(clrRed));
   canvas.CircleAA(x2,y2,3, ColorToARGB(clrBlue));
//--- 显示更新的canvas
   canvas.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

# Pie

在(x1,y1)和(x2,y2)位置的矩形上绘制椭圆形的填充扇形。直线从椭圆形中心横切扇形边界，延伸至(x3,y3)和(x4,y4)两个坐标点。

```
void  Pie(
   int          x1,       // 矩形左上角的 X 坐标
   int          y1,       // 矩形左上角的 Y 坐标
   int          x2,       // 矩形右下角的 X 坐标
   int          y2,       // 矩形右下角的 Y 坐标
   int          x3,       // 找到弧形边界的第一个点的 X 坐标
   int          y3,       // 找到弧形边界的第一个点的 Y 坐标
   int          x4,       // 找到弧形边界的第二个点的 X 坐标
   int          y4,       // 找到弧形边界的第二个点的 Y 坐标
   const uint   clr,      // 直线颜色
   const uint   fill_clr // 填充颜色
   );
```

参数

*x1*

[in] 形成矩形的左上角的 X 坐标。

*y1*

[in] 形成矩形的左上角的 Y 坐标。

*x2*

[in] 形成矩形的右下角的 X 坐标。

*y2*

[in] 形成矩形的右下角的 Y 坐标。

*x3*

[in] 直线从矩形中心点开始绘制到第一个点的 X 坐标，以获得弧形边界。

*y3*

[in] 直线从矩形中心点开始绘制到第一个点的 Y 坐标，以获得弧形边界。

*x4*

[in] 直线从矩形中心点开始绘制到第二个点的 X 坐标，以获得弧形边界。

*y4*

[in] 直线从矩形中心点开始绘制到第二个点的 Y 坐标，以获得弧形边界。

*clr*

[in] ARGB格式的扇形边界颜色。

*fill_clr*

[in] ARGB格式的扇形填充颜色。使用ColorToARGB()函数，将颜色转换为ARGB格式。

© 2000-2025, MetaQuotes Ltd.

在矩形上，绘制中心点在(x,y)，半径为rx和ry的椭圆形的填充扇形。使用通过fi3和fi4的夹角形成的射线从椭圆形中心横切扇形边界。

```
void  Pie(
   int          x,        // 椭圆形中心的 X 坐标
   int          y,        // 椭圆形中心的 Y 坐标
   int          rx,       // X 轴上的椭圆形半径
   int          ry,       // Y 轴上的椭圆形半径
   int          fi3,      // 从椭圆形中心发出的射线角，定义弧形的第一个边界
   int          fi4,      // 从椭圆形中心发出的射线角，定义弧形的第二个边界
   const uint   clr,      // 直线颜色
   const uint   fill_clr // 填充颜色
   );
```

在矩形上，绘制中心点在(x,y)，半径为rx和ry的椭圆形的填充扇形，并返回弧形边界的坐标。使用通过fi3和fi4的夹角形成的射线从椭圆形中心横切扇形边界。

```
void  Pie(
   int          x,        // 椭圆形中心的 X 坐标
   int          y,        // 椭圆形中心的 Y 坐标
   int          rx,       // X 轴上的椭圆形半径
   int          ry,       // Y 轴上的椭圆形半径
   int          fi3,      // 从椭圆形中心发出的射线角，定义弧形的第一个边界
   int          fi4,      // 从椭圆形中心发出的射线角，定义弧形的第二个边界
   int&         x3,       // 第一个弧形边界的 X 坐标
   int&         y3,       // 第一个弧形边界的 Y 坐标
   int&         x4,       // 第二个弧形边界的 X 坐标
   int&         y4,       // 第二个弧形边界的 Y 坐标
   const uint   clr,      // 直线颜色
   const uint   fill_clr // 填充颜色
   );
```

参数

*x*

[in] 椭圆形中心的 X 坐标。

*y*

[in] 椭圆形中心的 Y 坐标。

*rx*

[in] 像素表示的X轴上的椭圆形半径。

*ry*

[in] 像素表示的X轴上的椭圆形半径。

*fi3*

[in] 定义弧形第一个边界的弧度角。

*fi4*

[in] 定义弧形第二个边界的弧度角。

© 2000-2025, MetaQuotes Ltd.

*x3*

[out] 获得第一个弧形边界 X 坐标的变量。

*y3*

[out] 获得第一个弧形边界 Y 坐标的变量。

*x4*

[out] 获得第二个弧形边界 X 坐标的变量。

*y4*

[out] 获得第二个弧形边界 Y 坐标的变量。

*clr*

[in] ARGB格式的扇形边界颜色。

*fill_clr*

[in] ARGB格式的扇形填充颜色。使用ColorToARGB() 函数，将颜色转换为ARGB格式。

调用类函数的示例：

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int      Width=600;
   int      Height=400;
//--- 创建 canvas
   if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
     {
      Print("Error creating canvas: ",GetLastError());
     }
//--- 清除 canvas
   canvas.Erase(clrWhite);
//--- 绘制矩形
   canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- 绘制第一个饼图
   canvas.Pie(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrBlue),ColorToARGB(c
//--- 绘制第二个饼图
   canvas.Pie(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,ColorToARGB(clrGreen),Color
//--- 显示更新的canvas
   canvas.Update();
   DebugBreak();
  }
```

© 2000-2025, MetaQuotes Ltd.

# FillPolygon

绘制一个填充多边形。

```
void  FillPolygon(
   int&          x,       // 多边形各点的 X 坐标的数组
   int&          y,       // 多边形各点的 Y 坐标的数组
   const uint    clr      // 颜色
   );
```

参数

*x*

[in] 多边形各点的 X 坐标的数组。

*y*

[in] 多边形各点的 Y 坐标的数组。

*clr*

[in] ARGB颜色格式。

© 2000-2025, MetaQuotes Ltd.

# FillEllipse

在指定坐标的矩形上绘制填充椭圆形。

```
void  FillPolygon(
   int          x1,       // 矩形左上角的 X 坐标
   int          y1,       // 矩形左上角的 Y 坐标
   int          x2,       // 矩形右下角的 X 坐标
   int          y2,       // 矩形右下角的 Y 坐标
   const uint   clr        // 椭圆形颜色
   );
```

参数

*x1*

[in] 形成矩形的左上角的 X 坐标。

*y1*

[in] 形成矩形的左上角的 Y 坐标。

*x2*

[in] 形成矩形的右下角的 X 坐标。

*y2*

[in] 形成矩形的右下角的 Y 坐标。

*clr*

[in] ARGB颜色格式。

© 2000-2025, MetaQuotes Ltd.

# GetDefaultColor

根据索引编号返回预定义的颜色。

```
static uint  GetDefaultColor(
   const uint  i      // 索引编号
   );
```

参数

*i*

[in] 获得颜色的索引编号。

返回值

颜色。

© 2000-2025, MetaQuotes Ltd.

# ChartObjectName

获取一个绑定图表对象的名称。

```
string  ChartObjectName();
```

返回值

一个绑定图表对象的名称

© 2000-2025, MetaQuotes Ltd.

# Circle

绘制一个圆形

```
void  Circle(
   int          x,        // X 坐标
   int          y,        // Y 坐标
   int          r,        // 半径
   const uint   clr       // 颜色
   );
```

参数

*x*

[输入] 圆心的 X 坐标。

*y*

[输入] 圆心的 Y 坐标。

*r*

[输入] 圆形半径。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# CircleAA

使用反锯齿算法绘制一个圆形

```
void  CircleAA(
   const int      x,        // X 坐标
   const int      y,        // Y 坐标
   const double   r,        // 半径
   const uint     clr       // 颜色
   );
```

参数

*x*

[输入] 圆心的 X 坐标。

*y*

[输入] 圆心的 Y 坐标。

*r*

[输入] 圆形半径。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# CircleWu

Draws a circle using Wu's anti-aliasing algorithm

```
void  CircleWu(
   const int      x,        // X 坐标
   const int      y,        // Y 坐标
   const double   r,        // 半径
   const uint     clr       // 颜色
   );
```

参数

*x*

[输入] 圆心的 X 坐标。

*y*

[输入] 圆心的 Y 坐标。

*r*

[输入] 圆形半径。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# 创建

创建一个图形资源，未与图表对象绑定。

```
virtual bool  Create(
   const string       name,                                  // 名称
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 格式
   );
```

参数

*name*

[输入] 图形资源名称的前缀。在创建过程中资源名称通过添加一个伪随机字符串生成。

*width*

[输入] 宽度 (沿 X 轴尺寸)，单位为像素。

*height*

[输入] 高度 (沿 Y 轴尺寸)，单位为像素。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

[输入] 颜色处理方法。参阅 ResourceCreate() 函数描述来了解更多关于颜色处理的方法。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# CreateBitmap

创建一个图形资源，且与一个图表对象绑定。

1. 在当前图表的主窗口里创建一个图形资源。

```
bool  CreateBitmap(
   const string       name,                                  // 名称
   const datetime     time,                                  // 时间
   const double       price,                                 // 价位
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 格式
   );
```

2. 使用图表 ID 和子窗口号码创建一个图形资源。

```
bool  CreateBitmap(
   const long         chart_id,                              // 图表 ID
   const int          subwin,                                // 子窗口号码
   const string       name,                                  // 名称
   const datetime     time,                                  // 时间
   const double       price,                                 // 价位
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 格式
   );
```

参数

*chart_id*

[输入] 用于创建一个对象的图表 ID。

*subwin*

[输入] 用于创建一个对象的图表子窗口号码。

*name*

[输入] 图表对象名称和图形资源名称的前缀。

*time*

[输入] 图表对象锚点的时间坐标。

*price*

[输入] 图表对象锚点的价格坐标。

*width*

[输入] 图形资源宽度 (沿 X 轴尺寸)，单位为像素。

*height*

[输入] 图形资源高度 (沿 Y 轴尺寸)，单位为像素。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

© 2000-2025, MetaQuotes Ltd.

[输入] 颜色处理方法。参阅 ResourceCreate() 函数描述来了解更多关于颜色处理的方法。

返回值

true - 如果成功，否则 - false

注释

如果使用函数的第一种版本，则对象将被创建于当前图表的主窗口。

对象的大小将与图形资源的大小一致。

© 2000-2025, MetaQuotes Ltd.

# CreateBitmapLabel

创建一个图形资源，且与一个图表对象绑定。

1. 在当前图表的主窗口里创建一个图形资源。

```
bool  CreateBitmapLabel(
   const string       name,                                  // 名称
   const int          x,                                     // X 坐标
   const int          y,                                     // Y 坐标
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 格式
   );
```

2. 使用图表 ID 和子窗口号码创建一个图形资源。

```
bool  CreateBitmapLabel(
   const long         chart_id,                              // 图表 ID
   const int          subwin,                                // 子窗口号码
   const string       name,                                  // 名称
   const int          x,                                     // X 坐标
   const int          y,                                     // Y 坐标
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 格式
   );
```

参数

*chart_id*

[输入] 用于创建一个对象的图表 ID。

*subwin*

[输入] 用于创建一个对象的图表子窗口号码。

*name*

[输入] 图表对象名称和图形资源名称的前缀。

*x*

[输入] 图表对象锚点的 X 坐标。

*y*

[输入] 图表对象锚点的 Y 坐标。

*width*

[输入] 图形资源宽度 (沿 X 轴尺寸)，单位为像素。

*height*

[输入] 图形资源高度 (沿 Y 轴尺寸)，单位为像素。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

© 2000-2025, MetaQuotes Ltd.

[输入] 颜色处理方法。参阅 ResourceCreate() 函数描述来了解更多关于颜色处理的方法。

返回值

true - 如果成功，否则 - false

注释

如果使用函数的第一种版本，则对象将被创建于当前图表的主窗口。

对象的大小将与图形资源的大小一致。

© 2000-2025, MetaQuotes Ltd.

# 销毁

销毁一个图形资源。

```
void  Destroy();
```

注释

如果一个图形资源已经被绑定到一个图表对象，则后者将被删除。

© 2000-2025, MetaQuotes Ltd.

# Ellipse

Draws an ellipse using two points.

```
void  Ellipse(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   const uint   clr      // color
   );
```

**Parameters**

*x1*

[in] X coordinate of the first point forming an ellipse.

*y1*

[in] Y coordinate of the first point forming an ellipse.

*x2*

[in] X coordinate of the second point forming an ellipse.

*y2*

[in] Y coordinate of the second point forming an ellipse.

*clr*

[in] Color in ARGB format.

© 2000-2025, MetaQuotes Ltd.

# EllipseAA

Draws an ellipse based on two points using anti-aliasing algorithm.

```
void  EllipseAA(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   const uint   clr      // color
   );
```

**Parameters**

*x1*

[in] X coordinate of the first point forming an ellipse.

*y1*

[in] Y coordinate of the first point forming an ellipse.

*x2*

[in] X coordinate of the second point forming an ellipse.

*y2*

[in] Y coordinate of the second point forming an ellipse.

*clr*

[in] Color in ARGB format.

© 2000-2025, MetaQuotes Ltd.

# EllipseWu

Draws an ellipse based on two points using Wu's anti-aliasing algorithm.

```
void  EllipseWu(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

**Parameters**

*x1*

[in] X coordinate of the first point forming an ellipse.

*y1*

[in] Y coordinate of the first point forming an ellipse.

*x2*

[in] X coordinate of the second point forming an ellipse.

*y2*

[in] Y coordinate of the second point forming an ellipse.

*clr*

[in] Color in ARGB format.

© 2000-2025, MetaQuotes Ltd.

# Erase

擦除或填充指定颜色。

```
void  Erase(
   const uint  clr=0      // 颜色
   );
```

参数

*clr=0*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# Fill

填充一片区域。

```
void  Fill(
   int          x,        // X 坐标
   int          y,        // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x*

[输入] 填充起始点的 X 坐标。

*y*

[输入] 填充起始点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# FillCircle

绘制一个填充圆形。

```
void  FillCircle(
   int          x,        // X 坐标
   int          y,        // Y 坐标
   int          r,        // 半径
   const uint   clr       // 颜色
   );
```

参数

*x*

[输入] 填充圆心的 X 坐标。

*y*

[输入] 填充圆心的 Y 坐标。

*r*

[输入] 填充圆形的半径。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# FillRectangle

绘制一个填充长方形。

```
void  FillRectangle(
   int          x1,       // X 坐标
   int          y1,       // Y 坐标
   int          x2,       // X 坐标
   int          y2,       // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[输入]形成长方形的第一个点的 X 坐标。

*y1*

[输入]形成长方形的第一个点的 Y 坐标。

*x2*

[输入]形成长方形的第二个点的 X 坐标。

*y2*

[输入]形成长方形的第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# FillTriangle

绘制一个填充三角形。

```
void  FillTriangle(
   int          x1,       // X 坐标
   int          y1,       // Y 坐标
   int          x2,       // X 坐标
   int          y2,       // Y 坐标
   int          x3,       // X 坐标
   int          y3,       // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[输入] 三角形第一个对角的 X 坐标。

*y1*

[输入] 三角形第一个对角的 Y 坐标。

*x2*

[输入] 三角形第二个对角的 X 坐标。

*y2*

[输入] 三角形第二个对角的 Y 坐标。

*x3*

[输入] 三角形第三个边角的 X 坐标。

*y3*

[输入] 三角形第三个对角的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# FontAngleGet

获取字体倾斜角度。

```
uint  FontAngleGet();
```

返回值

字体倾斜角度

© 2000-2025, MetaQuotes Ltd.

# FontAngleSet

设置字体倾斜角度。

```
bool  FontAngleSet(
   uint  angle      // 角度
   );
```

参数

*angle*

[输入] 字体倾斜角度，单位为十分之一度。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# FontFlagsGet

获取字体标志。

```
uint  FontFlagsGet();
```

返回值

字体标志

© 2000-2025, MetaQuotes Ltd.

# FontFlagsSet

设置字体标志。

```
bool  FontFlagsSet(
   uint  flags      // 标志
   );
```

参数

*flags*

[输入] 字体创建标志。参阅 TextSetFont() 函数描述来了解更多关于标志。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# FontGet

获取当前字体参数。

```
void  FontGet(
   string&  name,       // 名称
   int&     size,       // 大小
   uint&    flags,      // 标志
   uint&    angle       // 倾斜角度
   );
```

参数

*name*

[输出] 返回字体名称变量的引用。

*size*

[输出] 返回字体大小变量的引用。

*flags*

[输出] 返回字体标志变量的引用。

*angle*

[输出] 返回字体倾斜角度变量的引用。

© 2000-2025, MetaQuotes Ltd.

# FontNameGet

获取字体名称。

```
string  FontNameGet();
```

返回值

字体名称

© 2000-2025, MetaQuotes Ltd.

# FontNameSet

设置字体名称。

```
bool  FontNameSet(
   string  name      // 名称
   );
```

参数

*name*

[输入] 字体名称。例如，"Arial"。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# FontSet

设置当前字体。

```
bool  FontSet(
   const string  name,         // 名称
   const int     size,         // 大小
   const uint    flags=0,      // 标志
   const uint    angle=0       // 角度
   );
```

参数

*name*

[输入] 字体名称。例如，"Arial"。

*size*

[输入] 字体大小。参阅 TextSetFont() 函数描述来了解更多关于大小设置。

*flags=0*

[输入] 字体创建标志。参阅 TextSetFont() 函数描述来了解更多关于标志。

*angle=0*

[输入] 字体倾斜角度，单位为十分之一度。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# FontSizeGet

获取字体大小。

```
int  FontSizeGet();
```

返回值

字体大小

© 2000-2025, MetaQuotes Ltd.

# FontSizeSet

设置字体大小。

```
bool  FontSizeSet(
   int  size      // 大小
   );
```

参数

*size*

[输入] 字体大小。参阅 TextSetFont() 函数描述来了解更多关于大小设置。

返回值

true - 如果成功，否则 - false

© 2000-2025, MetaQuotes Ltd.

# Height

获取图形资源的高度。

```
int  Height();
```

返回值

图形资源的高度

© 2000-2025, MetaQuotes Ltd.

# Line

绘制一条随意线段。

```
void  Line(
   int          x1,       // X 坐标
   int          y1,       // Y 坐标
   int          x2,       // X 坐标
   int          y2,       // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[输入] 线段第一个点的 X 坐标。

*y1*

[输入] 线段第一个点的 Y 坐标。

*x2*

[输入] 线段第二个点的 X 坐标。

*y2*

[输入] 线段第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# LineAA

使用反锯齿算法绘制一条手绘线。

```
void  LineAA(
   const int   x1,                  // X 坐标
   const int   y1,                  // Y 坐标
   const int   x2,                  // X 坐标
   const int   y2,                  // Y 坐标
   const uint  clr,                 // 颜色
   const uint  style=UINT_MAX       // 线形
   );
```

参数

*x1*

[输入] 线段第一个点的 X 坐标。

*y1*

[输入] 线段第一个点的 Y 坐标。

*x2*

[输入] 线段第二个点的 X 坐标。

*y2*

[输入] 线段第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# LineWu

Draws a segment of a freehand line using Wu's anti-aliasing algorithm.

```
void  LineWu(
   const int   x1,                  // X 坐标
   const int   y1,                  // Y 坐标
   const int   x2,                  // X 坐标
   const int   y2,                  // Y 坐标
   const uint  clr,                 // 颜色
   const uint  style=UINT_MAX       // 线形
   );
```

参数

*x1*

[输入] 线段第一个点的 X 坐标。

*y1*

[输入] 线段第一个点的 Y 坐标。

*x2*

[输入] 线段第二个点的 X 坐标。

*y2*

[输入] 线段第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# LineHorizontal

绘制一条水平线段。

```
void  LineHorizontal(
   int          x1,       // X 坐标
   int          x2,       // X 坐标
   int          y,        // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[输入] 线段第一个点的 X 坐标。

*x2*

[输入] 线段第二个点的 X 坐标。

*y*

[输入] 线段的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# LineVertical

绘制一条垂直线段。

```
void  LineVertical(
   int          x,        // X 坐标
   int          y1,       // Y 坐标
   int          y2,       // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x*

[输入] 线段的 X 坐标。

*y1*

[输入] 线段第一个点的 Y 坐标。

*y2*

[输入] 线段第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# LineStyleSet

设置线形。

```
void  LineStyleSet(
   const uint  style      // 线形
   );
```

参数

*style*

[输入] 线形。

注释

输入参数可以是任意 ENUM_LINE_STYLE 枚举数值。此外，可以创建一个自定义的绘制风格。

© 2000-2025, MetaQuotes Ltd.

# LineThick

通过抗锯齿算法绘制一段具有指定宽度的手绘线。

```
void  LineThick(
   const int       x1,            // 这段线条第一个点的X坐标
   const int       y1,            // 这段线条第一个点的Y坐标
   const int       x2,            // 这段线条第二个点的X坐标
   const int       y2,            // 这段线条第二个点的Y坐标
   const uint      clr,           // 颜色
   const int       size,          // 线条宽度
   const uint      style,         // 线条样式
   ENUM_LINE_END   end_style      // 线条末端样式
   )
```

参数

*x1*

[in] 这段线条第一个点的X坐标。

*y1*

[in] 这段线条第一个点的Y坐标。

*x2*

[in] 这段线条第二个点的X坐标。

*y2*

[in] 这段线条第二个点的Y坐标。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style*

[in] 线条样式就是ENUM_LINE_STYLE枚举值或自定义值的其中一个。

*end_style*

[in] 线条样式就是ENUM_LINE_END枚举值的其中一个

**ENUM_LINE_END**

| ID | 描述 |
| --- | --- |
| LINE_END_ROUND | 线条末端为圆形。 |
| LINE_END_BUTT | 线条末端剪切。 |
| LINE_END_SQUARE | 线条末端为填充好的矩形。 |

© 2000-2025, MetaQuotes Ltd.

# LineThickVertical

通过抗锯齿算法绘制一段具有指定宽度的手绘垂直线。

```
void  LineThickVertical(
   const int       x,              // 这段线条的X坐标
   const int       y1,             // 这段线条第一个点的Y坐标
   const int       y2,             // 这段线条第二个点的Y坐标
   const uint      clr,            // 颜色
   const int       size,           // 线条宽度
   const uint      style,          // 线条样式
   ENUM_LINE_END   end_style       // 线条末端样式
   )
```

参数

*x*

[in] 这段线条的X坐标。

*y1*

[in] 这段线条第一个点的Y坐标。

*y2*

[in] 这段线条第二个点的Y坐标。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style*

[in] 线条样式就是ENUM_LINE_STYLE 枚举值或自定义值的其中一个。

*end_style*

[in] 线条样式就是ENUM_LINE_END 枚举值的其中一个。

© 2000-2025, MetaQuotes Ltd.

# LineThickHorizontal

通过抗锯齿绘制一段具有指定宽度的手绘水平线。

```
void  LineThickHorizontal(
   const int       x1,             // 这段线条第一个点的X坐标
   const int       x2,             // 这段线条第二个点的X坐标
   const int       y,              // 这段线条的Y坐标
   const uint      clr,            // 颜色
   const int       size,           // 线条宽度
   const uint      style,          // 线条样式
   ENUM_LINE_END   end_style       // 线条末端样式
   )
```

参数

*x1*

[in] 这段线条第一个点的X坐标。

*x2*

[in] 这段线条第二个点的X坐标。

*y*

[in] 这段线条的Y坐标。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style*

[in] 线条样式就是ENUM_LINE_STYLE 枚举值或自定义值的其中一个。

*end_style*

[in] 线条样式就是ENUM_LINE_END 枚举值的其中一个。

© 2000-2025, MetaQuotes Ltd.

# LoadFromFile

从 BMP 文件里读取一个图像。

```
bool  LoadFromFile(
   const string  filename      // 文件名称
   );
```

参数

*filename*

[输入] 文件名称 (包含 "BMP" 扩展名)。

返回值

true - 如果成功, 否则 - false

© 2000-2025, MetaQuotes Ltd.

# PixelGet

获取指定坐标点的颜色。

```
uint  PixelGet(
   const int  x,      // X 坐标
   const int  y       // Y 坐标
   );
```

参数

*x*

[输入] 点的 X 坐标。

*y*

[输入] 点的 Y 坐标。

返回值

ARGB 格式的点颜色。

© 2000-2025, MetaQuotes Ltd.

# PixelSet

设置指定坐标点的颜色。

```
void  PixelSet(
   const int   x,        // X 坐标
   const int   y,        // Y 坐标
   const uint  clr       // 颜色
   );
```

参数

*x*

[输入] 点的 X 坐标。

*y*

[输入] 点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# PixelSetAA

使用反锯齿算法绘制一个点。

```
void  PixelSetAA(
   const double  x,        // X 坐标
   const double  y,        // Y 坐标
   const uint    clr       // 颜色
   );
```

参数

*x*

[输入] 点的 X 坐标。

*y*

[输入] 点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# Polygon

绘制一个多边形。

```
void  Polygon(
   int&         x[],      // X 坐标数组
   int&         y[],      // Y 坐标数组
   const uint   clr       // 颜色
   );
```

参数

*x[]*

[输入] 多边形顶点 X 坐标的数组。

*y[]*

[输入] 多边形顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# PolygonAA

使用反锯齿算法绘制一个多边形。

```
void  PolygonAA(
   int&         x[],                 // X 坐标数组
   int&         y[],                 // Y 坐标数组
   const uint   clr,                 // 颜色
   const uint   style=UINT_MAX       // 线形
   );
```

参数

*x[]*

[输入] 多边形顶点 X 坐标的数组。

*y[]*

[输入] 多边形顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# PolygonWu

Draws a polygon using Wu's anti-aliasing algorithm.

```
void  PolygonWu(
   int&         x[],                 // X 坐标数组
   int&         y[],                 // Y 坐标数组
   const uint   clr,                 // 颜色
   const uint   style=UINT_MAX       // 线形
   );
```

参数

*x[]*

[输入] 多边形顶点 X 坐标的数组。

*y[]*

[输入] 多边形顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# PolygonThick

通过抗锯齿算法绘制一个指定宽度的多边形。

```
void  PolygonThick(
   const int&       x[],          // 多边形各点的X坐标的数组
   const int&       y[],          // 多边形各点的 Y 坐标的数组
   const uint       clr,          // 颜色
   const int        size,         // 线条宽度
   const uint       style,        // 线条样式
   ENUM_LINE_END    end_style     // 线条末端样式
   )
```

参数

*x[]*

[in] 多边形各点的X坐标的数组。

*y[]*

[in] 多边形各点的 Y 坐标的数组。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style*

[in] 线条样式就是ENUM_LINE_STYLE 枚举值或自定义值的其中一个。

*end_style*

[in] 线条样式就是ENUM_LINE_END 枚举值的其中一个。

© 2000-2025, MetaQuotes Ltd.

# PolygonSmooth

通过两种抗锯齿算法连续绘制一个指定宽度的多边形。首先，单一线段是基于Bezier曲线平滑。然后，栅格抗锯齿算法应用于建自这些线段的多边形来提高渲染性。

```
void  PolygonSmooth(
   int&             x[],                    // 多边形各点的X坐标的数组
   int&             y[],                    // 多边形各点的 Y 坐标的数组
   const uint       clr,                    // 颜色
   const int        size,                   // 线条宽度
   ENUM_LINE_STYLE  style=STYLE_SOLID,      // 线条样式
   ENUM_LINE_END    end_style=LINE_END_ROUND,  // 线条末端样式
   double           tension=0.5,            // 抗锯齿参数值
   double           step=10                 // 近似线条的长度
   )
```

参数

*&x[]*

[in] 多边形各点的X坐标的数组。

*&y[]*

[in] 多边形各点的 Y 坐标的数组。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style=STYLE_SOLID*

[in] 线条样式就是ENUM_LINE_STYLE 枚举值或自定义值的其中一个。

*end_style=LINE_END_ROUND*

[in] 线条样式就是ENUM_LINE_END 枚举值的其中一个。

*tension=0.5*

[in] 平滑参数值。

*step=10*

[in] 近似线条的长度。

© 2000-2025, MetaQuotes Ltd.

# Polyline

绘制一条折线。

```
void  Polyline(
   int&         x[],      // X 坐标数组
   int&         y[],      // Y 坐标数组
   const uint   clr       // 颜色
   );
```

参数

*x[]*

[输入] 折线顶点 X 坐标的数组。

*y[]*

[输入] 折线顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# PolylineSmooth

通过两种抗锯齿算法连续绘制一个指定宽度的多段线。首先，单一线段是基于Bezier曲线平滑。然后，栅格抗锯齿算法应用于建自这些线段的多段线来提高渲染性。

```
void  PolylineSmooth(
   const int&        x[],                        // 多段线各点的X坐标的数组
   const int&        y[],                        // 多段线各点的 Y 坐标的数组
   const uint        clr,                        // 颜色
   const int         size,                       // 线条宽度
   ENUM_LINE_STYLE   style=STYLE_SOLID,          // 线条样式
   ENUM_LINE_END     end_style=LINE_END_ROUND,   // 线条末端样式
   double            tension=0.5,                // 抗锯齿参数值
   double            step=10                     // 近似步骤
   )
```

参数

*&x[]*

[in] 多段线的X坐标数组。

*&y[]*

[in] 多段线的Y坐标数组。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style=STYLE_SOLID*

[in] 线条样式就是ENUM_LINE_STYLE枚举值或自定义值的其中一个。

*end_style=LINE_END_ROUND*

[in] 线条样式就是ENUM_LINE_END枚举值的其中一个。

*tension=0.5*

[in] 平滑参数值。

*step=10*

[in] 近似步骤。

© 2000-2025, MetaQuotes Ltd.

# PolylineThick

通过抗锯齿算法绘制一个指定宽度的多段线。

```
void  PolylineThick(
   const int       &x[],          // 多段线各点的X坐标的数组
   const int       &y[],          // 多段线各点的 Y 坐标的数组
   const uint      clr,           // 颜色
   const int       size,          // 线条宽度
   const uint      style,         // 线条样式
   ENUM_LINE_END   end_style      // 线条末端样式
   )
```

参数

*&x[]*

[in] 多段线的X坐标数组。

*&y[]*

[in] 多段线的Y坐标数组。

*clr*

[in] ARGB颜色格式。

*size*

[in] 线条宽度。

*style*

[in] 线条样式就是ENUM_LINE_STYLE枚举值或自定义值的其中一个。

*end_style*

[in] 线条样式就是ENUM_LINE_END枚举值的其中一个

© 2000-2025, MetaQuotes Ltd.

# PolylineWu

Draws a polyline using Wu's anti-aliasing algorithm.

```
void  PolylineWu(
   int&         x[],                 // X 坐标数组
   int&         y[],                 // Y 坐标数组
   const uint   clr,                 // 颜色
   const uint   style=UINT_MAX       // 线形
   );
```

参数

*x[]*

[输入] 折线顶点 X 坐标的数组。

*y[]*

[输入] 折线顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# PolylineAA

使用反锯齿算法绘制一条折线。

```
void  PolylineAA(
   int&         x[],                 // X 坐标数组
   int&         y[],                 // Y 坐标数组
   const uint   clr,                 // 颜色
   const uint   style=UINT_MAX       // 线形
   );
```

参数

*x[]*

[输入] 折线顶点 X 坐标的数组。

*y[]*

[输入] 折线顶点 Y 坐标的数组。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# Rectangle

使用两顶点绘制一个长方形。

```
void  Rectangle(
   int          x1,       // X 坐标
   int          y1,       // Y 坐标
   int          x2,       // X 坐标
   int          y2,       // Y 坐标
   const uint   clr       // 颜色
   );
```

参数

*x1*

[输入]形成长方形的第一个点的 X 坐标。

*y1*

[输入]形成长方形的第一个点的 Y 坐标。

*x2*

[输入]形成长方形的第二个点的 X 坐标。

*y2*

[输入]形成长方形的第二个点的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# 调整大小

调整一个图形资源大小。

```
bool  Resize(
   const int  width,       // 宽度
   const int  height       // 高度
   );
```

参数

*width*

[输入] 图形资源的新宽度。

*height*

[输入] 图形资源的新高度。

返回值

true - 如果成功，否则 - false

注释

当调整大小时，以前的图像不会保存。

© 2000-2025, MetaQuotes Ltd.

# ResourceName

获取图形资源的名称。

```
string  ResourceName();
```

返回值

图形资源的名称

© 2000-2025, MetaQuotes Ltd.

# TextHeight

获取文本高度。

```
int  TextHeight(
   const string  text      // 文本
   );
```

参数

*text*

[输入] 测量文本。

返回值

文本高度，以像素为单位

注释

当前字体用于测量文本。

© 2000-2025, MetaQuotes Ltd.

# TextOut

显示文本。

```
void  TextOut(
   int          x,              // X 坐标
   int          y,              // Y 坐标
   string       text,           // 文本
   const uint   clr,            // 颜色
   uint         alignment=0     // 对齐
   );
```

参数

*x*

[输入] 文本锚点的 X 坐标。

*y*

[输入] 文本锚点的 Y 坐标。

*text*

[输入] 显示文本。

*clr*

[输入] ARGB 格式的颜色值。

*alignment=0*

[输入] 文本锚点方法。参阅 TextOut() 函数描述来了解更多关于锚点的方法。

注释

当前字体用于显示文本。

© 2000-2025, MetaQuotes Ltd.

# TextSize

获取文本大小。

```
void  TextSize(
   const string  text,        // 文本
   int&          width,       // 宽度
   int&          height       // 高度
   );
```

参数

*text*

[输入] 测量文本。

*width*

[输出] 返回文本大小变量的引用。

*height*

[输出] 返回文本高度变量的引用。

注释

当前字体用于测量文本。

© 2000-2025, MetaQuotes Ltd.

# TextWidth

获取文本宽度。

```
int  TextWidth(
   const string  text      // 文本
   );
```

参数

*text*

[输入] 测量文本。

返回值

文本高度，以像素为单位

注释

当前字体用于测量文本。

© 2000-2025, MetaQuotes Ltd.

# TransparentLevelSet

设置透明级别。

```
void  TransparentLevelSet(
   const uchar  value      // 数值
   );
```

参数

*value*

[输入] 透明级别的新值。

注释

0 代表完全透明，而 255 - 则是完全不透明。

设置透明级别将影响所有先前已绘部分。指定的透明级别不影响将来的构造。

© 2000-2025, MetaQuotes Ltd.

# Triangle

绘制一个三角形。

```
void  Triangle(
   int          x1,      // X 坐标
   int          y1,      // Y 坐标
   int          x2,      // X 坐标
   int          y2,      // Y 坐标
   int          x3,      // X 坐标
   int          y3,      // Y 坐标
   const uint   clr      // 颜色
   );
```

参数

*x1*

[输入] 三角形第一个对角的 X 坐标。

*y1*

[输入] 三角形第一个对角的 Y 坐标。

*x2*

[输入] 三角形第二个对角的 X 坐标。

*y2*

[输入] 三角形第二个对角的 Y 坐标。

*x3*

[输入] 三角形第三个边角的 X 坐标。

*y3*

[输入] 三角形第三个对角的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

© 2000-2025, MetaQuotes Ltd.

# TriangleAA

使用反锯齿算法绘制一个三角形。

```
void  TriangleAA(
   const int   x1,                  // X 坐标
   const int   y1,                  // Y 坐标
   const int   x2,                  // X 坐标
   const int   y2,                  // Y 坐标
   const int   x3,                  // X 坐标
   const int   y3,                  // Y 坐标
   const uint  clr,                 // 颜色
   const uint  style=UINT_MAX       // 线形
   );
```

参数

*x1*

[输入] 三角形第一个对角的 X 坐标。

*y1*

[输入] 三角形第一个对角的 Y 坐标。

*x2*

[输入] 三角形第二个对角的 X 坐标。

*y2*

[输入] 三角形第二个对角的 Y 坐标。

*x3*

[输入] 三角形第三个边角的 X 坐标。

*y3*

[输入] 三角形第三个对角的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# TriangleWu

Draws a triangle using Wu's anti-aliasing algorithm.

```
void  TriangleWu(
   const int   x1,                  // X 坐标
   const int   y1,                  // Y 坐标
   const int   x2,                  // X 坐标
   const int   y2,                  // Y 坐标
   const int   x3,                  // X 坐标
   const int   y3,                  // Y 坐标
   const uint  clr,                 // 颜色
   const uint  style=UINT_MAX       // 线形
   );
```

参数

*x1*

[输入] 三角形第一个对角的 X 坐标。

*y1*

[输入] 三角形第一个对角的 Y 坐标。

*x2*

[输入] 三角形第二个对角的 X 坐标。

*y2*

[输入] 三角形第二个对角的 Y 坐标。

*x3*

[输入] 三角形第三个边角的 X 坐标。

*y3*

[输入] 三角形第三个对角的 Y 坐标。

*clr*

[输入] ARGB 格式的颜色值。

*style=UINT_MAX*

[输入] 线形是 ENUM_LINE_STYLE 枚举值之一或一个自定义数值。

© 2000-2025, MetaQuotes Ltd.

# Update

在屏幕上显示变化。

```
void  Update(
   const bool  redraw=true      // 标志
   );
```

参数

*redraw=true*

图表有必要重绘的标志。

© 2000-2025, MetaQuotes Ltd.

# Width

获取图形资源的宽度。

```
int  Width();
```

返回值

图形资源宽度

© 2000-2025, MetaQuotes Ltd.

# CChartCanvas

实施用于绘制图表及其元素的类的基类。

## 描述

该类包括处理任何图表基础元素的方法：坐标轴及其标志，图表图例，网格，背景等等。在这里您可以自定义显示元素的选项：可见性，文本颜色等等。

## 声明

```
class CChartCanvas : public CCanvas
```

## 主题

```
#include <Canvas\Charts\ChartCanvas.mqh>
```

## 继承体系

CCanvas

CChartCanvas

直接分支

CHistogramChart, CLineChart, CPieChart

## 类函数

| 方法 | 动作 |
| --- | --- |
| ColorBackground | 返回和设置背景颜色。 |
| ColorBorder | 返回和设置边框颜色。 |
| ColorText | 返回和设置文本颜色。 |
| ColorGrid | 返回和设置网格颜色。 |
| MaxData | 返回和设置可允许的最大数据量（序列）。 |
| MaxDescrLen | 返回和设置描述符的最大长度。 |
| ShowFlags | 返回和设置图表元素的可视标识。 |
| IsShowLegend | 返回和设置图表上的图例可视标识。 |
| IsShowScaleLeft | 返回左侧比例值的可视标识。 |
| IsShowScaleRight | 返回右侧比例值的可视标识。 |
| IsShowScaleTop | 返回顶部比例值的可视标识。 |
| IsShowScaleBottom | 返回底部比例值的可视标识。 |
| IsShowGrid | 返回图表网格的可视标识。 |
| IsShowDescriptors | 返回图表描述符的可视标识。 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 动作 |
| --- | --- |
| IsShowPercent | 返回图表百分比的可视标识。 |
| VScaleMin | 返回和设置纵坐标最小值。 |
| VScaleMax | 返回和设置纵坐标最大值。 |
| NumGrid | 返回和设置绘制图表网格时的纵坐标标尺数。 |
| DataOffset | 返回和设置数据偏移值。 |
| DataTotal | 返回图表数据序列的总数。 |
| DrawDescriptors | 绘制描述符的虚拟方法。 |
| DrawData | 在指定索引绘制数据序列的虚拟方法。 |
| Create | 创建图形资源的虚拟方法。 |
| AllowedShowFlags | 设置一组可允许的图表元素可视标识。 |
| ShowLegend | 设置图例的可视标识。 |
| ShowScaleLeft | 设置左侧比例的可视标识。 |
| ShowScaleRight | 设置右侧比例的可视标识。 |
| ShowScaleTop | 设置顶部比例的可视标识。 |
| ShowScaleBottom | 设置底部比例的可视标识。 |
| ShowGrid | 设置网格的可视标识。 |
| ShowDescriptors | 设置描述符的可视标识。 |
| ShowValue | 设置值的可视标识。 |
| ShowPercent | 设置百分比的可视标识。 |
| LegendAlignment | 设置图例的文本对齐方式。 |
| Accumulative | 设置序列的值累计标识。 |
| VScaleParams | 设置纵坐标值的参数。 |
| DescriptorUpdate | 更新序列描述符的值（指定位置）。 |
| ColorUpdate | 更新序列颜色（指定位置）。 |
| ValuesCheck | 执行绘制图表的内部计算。 |
| Redraw | 重新绘制图表。 |
| DrawBackground | 绘制背景。 |
| DrawLegend | 重新绘制图例。 |

© 2000-2025, MetaQuotes Ltd.

| 方法 | 动作 |
| --- | --- |
| DrawLegendVertical | 绘制垂直图例。 |
| DrawLegendHorizontal | 绘制水平图例。 |
| CalcScales | 计算比例坐标。 |
| DrawScales | 重新绘制全部值比例。 |
| DrawScaleLeft | 重新绘制左侧值比例。 |
| DrawScaleRight | 重新绘制右侧值比例。 |
| DrawScaleTop | 重新绘制顶部值比例。 |
| DrawScaleBottom | 重新绘制底部值比例。 |
| DrawGrid | 重新绘制图表。 |
| DrawChart | 重新绘制图表。 |

方法继承自类 CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

© 2000-2025, MetaQuotes Ltd.

## ColorBackground (获得方法)

返回背景色。

```
uint  ColorBackground()
```

返回值

Background color.

## ColorBackground (设置方法)

设置背景色。

```
void  ColorBackground(
  const uint  value,  //背景色
  )
```

参数

*value*

[in] 背景色。

© 2000-2025, MetaQuotes Ltd.

## ColorBorder (获得方法)

返回边框颜色。

```
uint  ColorBorder()
```

返回值

边框颜色。

## ColorBorder (设置方法)

设置边框颜色。

```
void  ColorBorder(
  const uint  value,  //边框颜色
  )
```

参数

*value*

[in] 边框颜色。

© 2000-2025, MetaQuotes Ltd.

## ColorText (获得方法)

返回文本颜色。

```
uint  ColorText()
```

返回值

文本颜色。

## ColorText (设置方法)

设置文本颜色。

```
void  ColorText(
  const uint  value,  // 文本颜色
  )
```

参数

*value*

[in] 文本颜色。

© 2000-2025, MetaQuotes Ltd.

## ColorGrid (获得方法)

返回网格颜色。

```
uint  ColorGrid()
```

返回值

网格颜色。

## ColorGrid (设置方法)

设置网格颜色。

```
void  ColorGrid(
  const uint  value,  // 网格颜色
  )
```

参数

*value*

[in] 网格颜色。

© 2000-2025, MetaQuotes Ltd.

## MaxData (获得方法)

返回可允许的最大数据量 (序列)。

```
uint  MaxData()
```

返回值

最大数据量 (序列)。

## MaxData (设置方法)

设置可允许的最大数据量 (序列)。

```
void  MaxData(
  const uint  value,  // 数据量
  )
```

参数

*value*

[in] 最大数据量 (序列)。

© 2000-2025, MetaQuotes Ltd.

## MaxDescrLen（获得方法）

返回描述符的最大长度。

```
uint  MaxDescrLen()
```

返回值

描述符的最大长度值。

## MaxDescrLen（设置方法）

设置描述符的最大长度。

```
void  MaxDescrLen(
  const uint  value,  // 最大长度
  )
```

参数

*value*

[in] 描述符的最大长度值。

© 2000-2025, MetaQuotes Ltd.

## ShowFlags（获得方法）

返回图表元素的可视标识。

```
bool  ShowFlags()
```

返回值

图表元素的可视标识值。

## ShowFlags（设置方法）

设置图表元素的可视标识。

```
void  ShowFlags(
  const uint  flags,  // 标识
  )
```

参数

*标识*

[in] 图表元素的可视标识值。

© 2000-2025, MetaQuotes Ltd.

# IsShowLegend

返回和设置图表上的图例可视标识。

```
bool  IsShowLegend()
```

返回值

如果图例可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowScaleLeft

返回左侧比例值的可视标识。

```
bool  IsShowScaleLeft()
```

返回值

如果比例值可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowScaleRight

返回右侧比例值的可视标识。

```
bool  IsShowScaleRight()
```

返回值

如果比例值可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowScaleTop

返回顶部比例值的可视标识。

```
bool  IsShowScaleTop()
```

返回值

如果比例值可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowScaleBottom

返回底部比例值的可视标识。

```
bool  IsShowScaleBottom()
```

返回值

如果比例值可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowGrid

返回图表网格的可视标识。

```
bool  IsShowGrid()
```

返回值

如果网格可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowDescriptors

返回图表描述符的可视标识。

```
bool  IsShowDescriptors()
```

返回值

如果描述符可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# IsShowPercent

返回图表百分比的可视标识。

```
bool  IsShowPercent()
```

返回值

如果百分比可见则返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

## VScaleMin (Get method)

返回纵坐标最小值。

```
double  VScaleMin()
```

返回值

纵坐标最小值。

## VScaleMin (设置方法)

设置纵坐标最小值。

```
void  VScaleMin(
  const double  value,    // 纵坐标值
  )
```

参数

*value*

[in] 最小值。

© 2000-2025, MetaQuotes Ltd.

## VScaleMax

返回纵坐标最大值。

```
double  VScaleMax()
```

返回值

纵坐标最大值。

## VScaleMax

设置纵坐标最大值。

```
void  VScaleMax(
  const double  value,  //   纵坐标值
  )
```

参数

*value*

[in] 最大值。

© 2000-2025, MetaQuotes Ltd.

## NumGrid

返回绘制图表网格时的纵坐标标尺数。

```
uint  NumGrid()
```

返回值

标尺数。

## NumGrid

设置绘制图表网格时的纵坐标标尺数。

```
void  NumGrid(
  const uint  value,  // 标尺数
  )
```

参数

*value*

[in] 标尺数。

© 2000-2025, MetaQuotes Ltd.

## DataOffset

返回数据偏移值。

```
int  DataOffset()
```

返回值

数据偏移。

## DataOffset

设置数据偏移值。

```
void  DataOffset(
  const int  value,  //偏移
  )
```

参数

*value*

[in] 数据偏移。

© 2000-2025, MetaQuotes Ltd.

# DataTotal

返回图表数据序列的总数。

```
uint  DataTotal()
```

返回值

序列号。

© 2000-2025, MetaQuotes Ltd.

# DrawDescriptors

绘制描述符的虚拟方法。

```
virtual void  DrawDescriptors()
```

© 2000-2025, MetaQuotes Ltd.

# DrawData

在指定索引绘制数据序列的虚拟方法。

```
virtual void  DrawData(
   const uint  idx=0,  // 索引编号
   )
```

参数

*idx=0*

[in] 序列索引。

© 2000-2025, MetaQuotes Ltd.

# Create

创建图形资源的虚拟方法。

```
virtual bool  Create(
  const string       name,      // 资源名称
  const int          width,     // 宽度
  const int          height,    // 高度
  ENUM_COLOR_FORMAT  clrfmt,    // 格式
  )
```

参数

*名称*

[in] 图形资源名称的基础。资源名称是在添加伪随机字符串创建时生成的。

*宽度*

[in] 以像素为单位的宽度（沿X轴的大小）。

*高度*

[in] 以像素为单位的高度（沿Y轴的大小）。

*clrfmt*

[in] 颜色处理方法。查看ResourceCreate()函数描述来了解更多有关颜色处理方法。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# AllowedShowFlags

设置一组可允许的图表元素可视标识。

```
void  AllowedShowFlags(
  const uint  flags,  // 标识
  )
```

参数

*标识*

[in] 可允许标识。

© 2000-2025, MetaQuotes Ltd.

# ShowLegend

设置图例的可视标识值(FLAG_SHOW_LEGEND)。

```
void  ShowLegend(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true — 图例可见。
- false —图例不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowScaleLeft

设置左侧比例的可视标识值(FLAG_SHOW_SCALE_LEFT)。

```
void  ShowScaleLeft(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —左侧比例可见。
- false —左侧比例不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowScaleRight

设置右侧比例的可视标识值(FLAG_SHOW_SCALE_RIGHT)。

```
void  ShowScaleRight(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true — 右侧比例可见。
- false —右侧比例不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowScaleTop

设置顶部比例的可视标识值 (FLAG_SHOW_SCALE_TOP)。

```
void  ShowScaleTop(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in] 标识值：

- true — 顶部比例可见。
- false —顶部比例不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowScaleBottom

设置底部比例的可视标识值(FLAG_SHOW_SCALE_BOTTOM)。

```
void  ShowScaleBottom(
  const bool  flag,     // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —底部比例可见。
- false —底部比例不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowGrid

设置网格的可视标识值(FLAG_SHOW_GRID)。

```
void  ShowGrid(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —网格可见。
- false —网格不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowDescriptors

设置描述符的可视标识值(FLAG_SHOW_DESCRIPTORS)。

```
void  ShowDescriptors(
  const bool  flag,    // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —描述符可见。
- false —描述符不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowValue

设置值的可视标识(FLAG_SHOW_VALUE)。

```
void  ShowValue(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —值可见。
- false —值不可见。

© 2000-2025, MetaQuotes Ltd.

# ShowPercent

设置百分比的可视标识值(FLAG_SHOW_PERCENT)。

```
void  ShowPercent(
  const bool  flag,  // 标识值
  )
```

参数

*flag*

[in]标识值：

- true —百分比可见。
- false —百分比不可见。

© 2000-2025, MetaQuotes Ltd.

# LegendAlignment

设置图例的文本对齐方式。

```
void  LegendAlignment(
  const ENUM_ALIGNMENT  value,  // 标识
  )
```

参数

*value*

[in] 获得ENUM_ALIGNMENT枚举值之一：

- ALIGNMENT_LEFT —左对齐。
- ALIGNMENT_TOP —顶部对齐。
- ALIGNMENT_RIGHT —右对齐。
- ALIGNMENT_BOTTOM —底部对齐。

© 2000-2025, MetaQuotes Ltd.

# Accumulative

设置序列的值累计标识。

```
void  Accumulative(
  const bool  flag=true,  // 标识值
  )
```

参数

*flag=true*

[in] 标识值：

- true — 当前序列值由之前全部值的总和来代替。
- false — 绘制序列的标准模式。

© 2000-2025, MetaQuotes Ltd.

# VScaleParams

设置纵坐标值的参数。

```
void  VScaleParams(
  const double  max,   // 最大值
  const double  min,   // 最小值
  const uint    grid,  // 标尺数
  )
```

参数

*max*

[in] 最小值。

*min*

[in] 最大值。

*grid*

[in] 比例尺数。

© 2000-2025, MetaQuotes Ltd.

# DescriptorUpdate

更新序列描述符的值（指定位置）。

```
bool  DescriptorUpdate(
  const uint    pos,    // 索引编号
  const string  descr,  //值
  )
```

参数

*pos*

[in] 序列索引 —从0开始的添加序列号。

*descr*

[in] 描述符值。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ColorUpdate

更新序列颜色（指定位置）。

```
bool  ColorUpdate(
  const uint  pos,  //索引编号
  const uint  clr,  //颜色
  )
```

参数

*pos*

[in] 序列索引 — 从0开始的添加序列号。

*clr*

[in] 颜色值。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValuesCheck

辅助虚拟方法，执行绘制图表的内部计算。

```
virtual void  ValuesCheck()
```

© 2000-2025, MetaQuotes Ltd.

# Redraw

重新绘制图表的虚拟方法。

```
virtual void  Redraw()
```

© 2000-2025, MetaQuotes Ltd.

# DrawBackground

重新绘制背景的虚拟方法。

```
virtual void  DrawBackground()
```

© 2000-2025, MetaQuotes Ltd.

# DrawLegend

重新绘制图例的虚拟方法。

```
virtual void  DrawLegend()
```

© 2000-2025, MetaQuotes Ltd.

# DrawLegendVertical

绘制垂直图例。

```
int  DrawLegendVertical(
  const int  w,  // 宽度
  const int  h,  // 高度
  )
```

参数

*w*

[in] 图例中文本的最大宽度。

*h*

[in] 图例中文本的最大高度。

返回值

以像素为单位图例的宽度。

© 2000-2025, MetaQuotes Ltd.

# DrawLegendHorizontal

绘制水平图例。

```
int  DrawLegendHorizontal(
  const int  w,  //
  const int  h,  //
  )
```

参数

*w*

[in] 图例中文本的最大宽度。

*h*

[in] 图例中文本的最大高度。

返回值

以像素为单位图例的高度。

© 2000-2025, MetaQuotes Ltd.

# CalcScales

计算标签坐标值比例的虚拟方法。

```
virtual void  CalcScales()
```

© 2000-2025, MetaQuotes Ltd.

# DrawScales

重新绘制全部比例值的虚拟方法。

```
virtual void  DrawScales()
```

© 2000-2025, MetaQuotes Ltd.

# DrawScaleLeft

重新绘制左侧比例值的虚拟方法。

```
virtual int  DrawScaleLeft(
  const bool  draw,  // 标识
  )
```

参数

*draw*

[in] 表示比例是否需要重新绘制的标识。

返回值

比例值的宽度。

© 2000-2025, MetaQuotes Ltd.

# DrawScaleRight

重新绘制右侧比例值的虚拟方法。

```
virtual int  DrawScaleRight(
  const bool  draw,  // 标识
  )
```

参数

*draw*

[in] 表示比例是否需要重新绘制的标识。

返回值

比例值的宽度。

© 2000-2025, MetaQuotes Ltd.

# DrawScaleTop

重新绘制顶部比例值的虚拟方法。

```
virtual int  DrawScaleTop(
  const bool  draw,  // 标识
  )
```

参数

*draw*

[in] 表示比例是否需要重新绘制的标识。

返回值

比例值的高度。

© 2000-2025, MetaQuotes Ltd.

# DrawScaleBottom

重新绘制底部比例值的虚拟方法。

```
virtual int  DrawScaleBottom(
  const bool  draw,  // 标识
  )
```

参数

*draw*

[in] 表示比例是否需要重新绘制的标识。

返回值

比例值的高度。

© 2000-2025, MetaQuotes Ltd.

# DrawGrid

重新绘制网格的虚拟方法。

```
virtual void  DrawGrid()
```

© 2000-2025, MetaQuotes Ltd.

# DrawChart

重新绘制图表的虚拟方法。

```
virtual void  DrawChart()
```

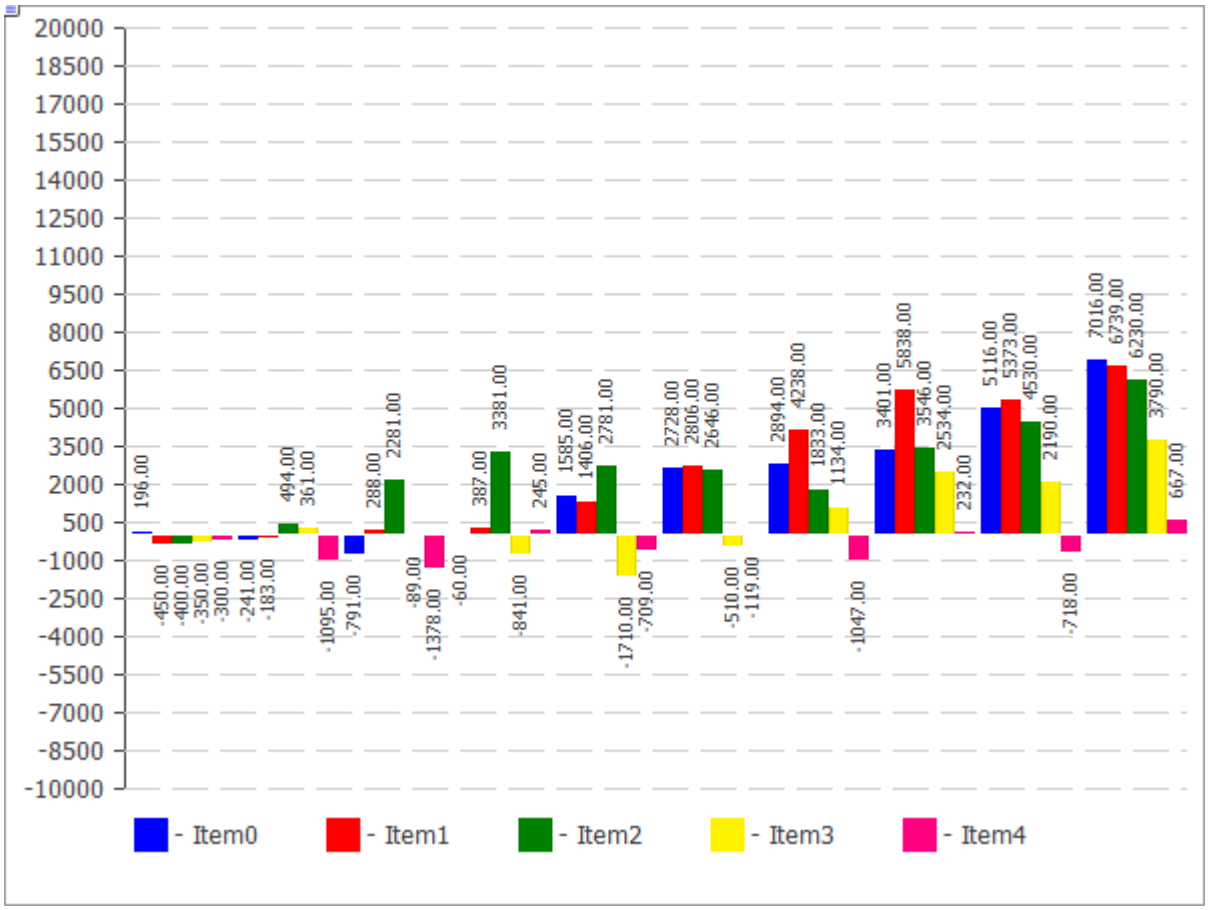
© 2000-2025, MetaQuotes Ltd.

# CHistogramChart

绘制直方图的类。

## 描述

在这个类中实施使用绘制直方图的全部方法。它们可以用于设置列宽和配置数据序列。包括使用逐渐填充直方图的方法，使数据更清晰的可视化。




下面提供图形上方的代码。

## 声明

```
class CHistogramChart : public CChartCanvas
```

## 主题

```
#include <Canvas\Charts\HistogramChart.mqh>
```

## 继承体系

CCanvas

CChartCanvas

CHistogramChart

## 类函数

© 2000-2025, MetaQuotes Ltd.

| 方法 | 动作 |
| --- | --- |
| Gradient | 设置显示是否使用逐渐填充直方图列的标识。 |
| BarGap | 设置自原点的直方图偏移值。 |
| BarMinSize | 设置直方图的最小列宽。 |
| BarBorder | 设置需要为每个列绘制边框的标识。 |
| Create | 创建图形资源的虚拟方法。 |
| SeriesAdd | 添加新的数据序列。 |
| SeriesInsert | 插入数据序列到图表。 |
| SeriesUpdate | 更新图表上的数据序列。 |
| SeriesDelete | 从图表删除数据序列。 |
| ValueUpdate | 更新指定序列的元素值。 |
| DrawData | 为指定序列绘制直方图的虚拟方法。 |
| DrawBar | 将直方图列绘制为填充矩形。 |
| GradientBrush | 为逐渐填充创建笔刷。 |

方法继承自类CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

方法继承自类CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags, ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

示例

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                          HistogramChartSample.mq5 |
//|                  Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Example of using histogram"
//---
#include <Canvas\Charts\HistogramChart.mqh>
//+------------------------------------------------------------------+
//| 输入                                                             |
//+------------------------------------------------------------------+
input bool Accumulative=true;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   int k=100;
   double arr[10];
//--- 创建图表
   CHistogramChart chart;
   if(!chart.CreateBitmapLabel("SampleHistogramChart",10,10,600,450))
     {
      Print("Error creating histogram chart: ",GetLastError());
      return(-1);
     }
   if(Accumulative)
     {
      chart.Accumulative();
      chart.VScaleParams(20*k*10,-10*k*10,20);
     }
   else
      chart.VScaleParams(20*k,-10*k,20);
   chart.ShowValue(true);
   chart.ShowScaleTop(false);
   chart.ShowScaleBottom(false);
   chart.ShowScaleRight(false);
   chart.ShowLegend();
   for(int j=0;j<5;j++)
     {
      for(int i=0;i<10;i++)
        {
         k=-k;
         if(k>0)
            arr[i]=k*(i+10-j);
         else
            arr[i]=k*(i+10-j)/2;
        }
      chart.SeriesAdd(arr,"Item"+IntegerToString(j));
     }
//--- 使用值
   while(!IsStopped())
     {
      int i=rand()%5;
      int j=rand()%10;
      k=rand()%3000-1000;
      chart.ValueUpdate(i,j,k);
      Sleep(200);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 完成
   chart.Destroy();
   return(0);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Gradient

设置显示是否使用逐渐填充直方图列的标识。

```
void  Gradient(
  const bool  flag=true,  // 标识值
  )
```

参数

*flag=true*

标识值：如果启用逐渐填充返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# BarGap

设置自原点的直方图偏移值。

```
void  BarGap(
  const uint  value,  // 偏移
  )
```

参数

*value*

[in] 直方图偏移值。

© 2000-2025, MetaQuotes Ltd.

# BarMinSize

设置直方图的最小列宽。

```
void  BarMinSize(
  const uint  value,  // 最小宽度
  )
```

参数

*value*

[in] 最小宽度。

© 2000-2025, MetaQuotes Ltd.

# BarBorder

设置需要为每个列绘制边框的标识。

```
void  BarBorder(
  const uint  value,  // 标识
  )
```

参数

*value*

[in] 标识值：

- true —将绘制边框
- false —不绘制边框

© 2000-2025, MetaQuotes Ltd.

# Create

创建图形资源的虚拟方法。

```
virtual bool  Create(
  const string       name,    //名称
  const int          width,   //宽度
  const int          height,  //高度
  ENUM_COLOR_FORMAT  clrfmt,  //格式
  )
```

参数

*名称*

[in] 图形资源名称的基础。资源名称是在添加伪随机字符串创建时生成的。

*宽度*

[in] 以像素为单位的宽度（沿X轴的大小）。

*高度*

[in] 以像素为单位的高度（沿Y轴的大小）。

*clrfmt*

[in] 颜色处理方法。查看ResourceCreate()函数描述来了解更多有关颜色处理方法。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesAdd

添加新的数据序列。

```
bool  SeriesAdd(
  const double&  value[],  // 值
  const string   descr,    // 标签
  const uint     clr,      // 颜色
  )
```

参数

*value[]*

[in] 数据序列。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesInsert

插入数据序列到图表。

```
bool  SeriesInsert(
  const uint    pos,        //索引编号
  const double&  value[],  // 值
  const string  descr,      // 标签
  const uint    clr,        // 颜色
  )
```

参数

*pos*

[in] 插入索引。

*value[]*

[in] 数据序列。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesUpdate

图表上的更新数据序列。

```
bool  SeriesUpdate(
  const uint    pos,        //索引编号
  const double  &value[],   //值
  const string  descr,      // 标签
  const uint    clr,        // 颜色
  )
```

参数

*pos*

[in] 序列索引 —从0开始的添加序列号。

*&value[]*

[in] 数据序列的新值。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则false。

© 2000-2025, MetaQuotes Ltd.

# SeriesDelete

从图表删除数据序列。

```
bool  SeriesDelete(
  const uint  pos,  //索引编号
  )
```

参数

*pos*

[in] 序列索引 — 从0开始的添加序列号。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueUpdate

更新指定序列的指定值。

```
bool  ValueUpdate(
  const uint   series,  // 序列索引
  const uint   pos,     // 元素索引
  double       value,   // 值
  )
```

参数

*series*

[in] 序列索引 —从0开始的添加序列号。

*pos*

[in] 序列元素索引。

*value*

[in] 新值。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# DrawData

为指定序列绘制直方图的虚拟方法。

```
virtual void  DrawData(
  const uint  index,  // 索引编号
  )
```

参数

*index*

[in] 序列索引 —从0开始的添加序列号。

© 2000-2025, MetaQuotes Ltd.

# DrawBar

将直方图列绘制为填充矩形。

```
void  DrawBar(
  const int   x,    // X坐标
  const int   y,    // Y坐标
  const int   w,    // 宽度
  const int   h,    //高度
  const uint  clr,  //颜色
  )
```

参数

*x*

[in] 矩形左上点的X坐标。

*y*

[in] 矩形左上点的Y坐标。

*w*

[in] 矩形的宽度。

*h*

[in] 矩形的高度。

*clr*

[in] 矩形的颜色。

© 2000-2025, MetaQuotes Ltd.

# GradientBrush

为逐渐填充创建笔刷。

```
void  GradientBrush(
  const int   size,      // 大小
  const uint  fill_clr,  // 填充颜色
  )
```

参数

*size*

[in] 笔刷强度

*fill_clr*

[in] 填充颜色

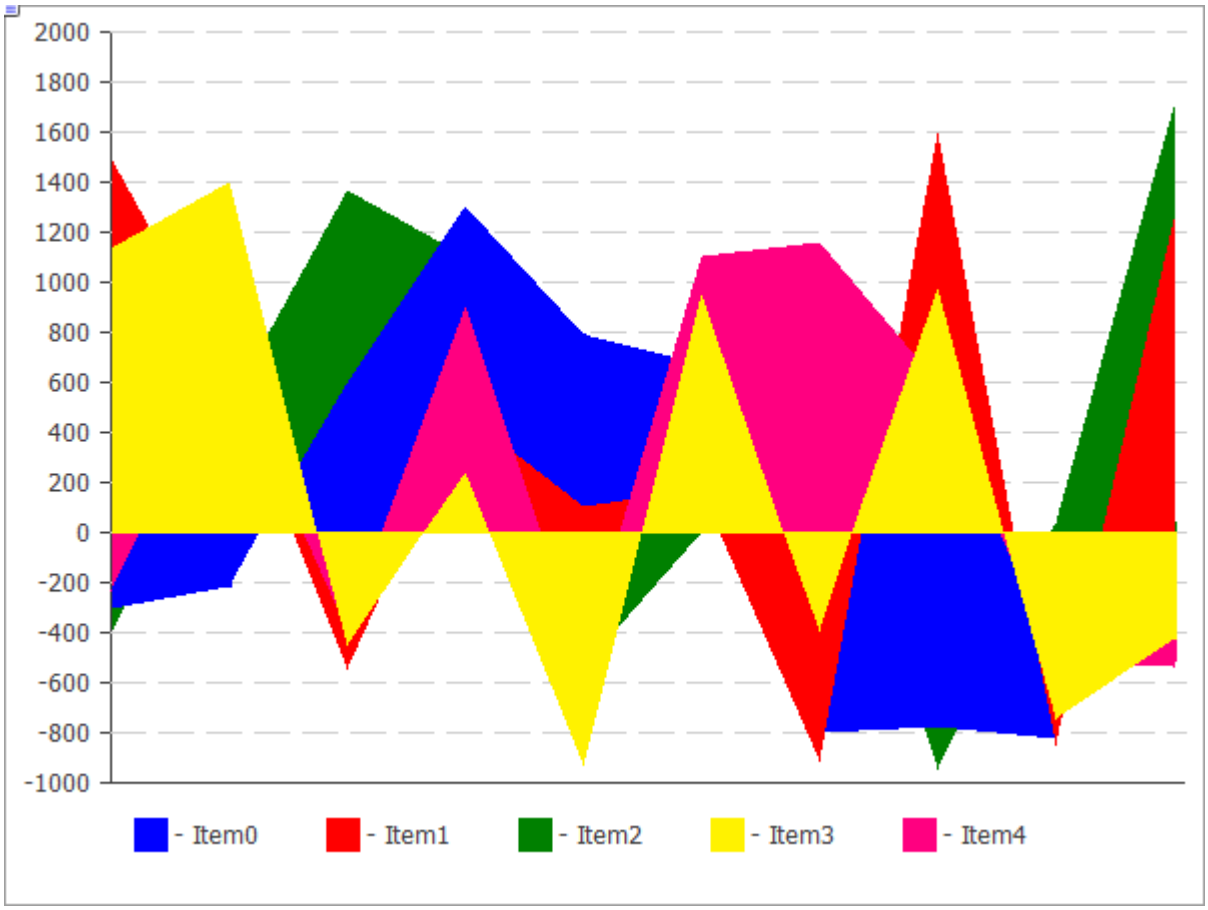
© 2000-2025, MetaQuotes Ltd.

# CLineChart

绘制曲线的类。

## 描述

该类中包含的方法专为图表上的曲线而设计。它具有填充绘制曲线限定内的区域的能力。




下面提供图形上方的代码。

## 声明

```
class CLineChart : public CChartCanvas
```

## 主题

```
#include <Canvas\Charts\LineChart.mqh>
```

## 继承体系

CCanvas

CChartCanvas

CLineChart

## 类函数

© 2000-2025, MetaQuotes Ltd.

| 方法 | 动作 |
| --- | --- |
| Filled | 设置填充数据序列定义曲线下的区域的标识。 |
| Create | 创建图形资源。 |
| SeriesAdd | 添加新的数据序列。 |
| SeriesInsert | 插入数据序列到图表。 |
| SeriesUpdate | 更新图表上的数据序列。 |
| SeriesDelete | 从图表删除数据序列。 |
| ValueUpdate | 更新指定序列的指定值。 |
| DrawChart | 绘制曲线及其所有元素的虚拟方法。 |
| DrawData | 为指定序列绘制曲线的虚拟方法。 |
| CalcArea | 计算数据序列定义曲线下的区域。 |

方法继承自类CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

方法继承自类CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags, ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

示例

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                LineChartSample.mq5 |
//|                  Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Example of using line chart"
//---
#include <Canvas\Charts\LineChart.mqh>
//+------------------------------------------------------------------+
//| 输入                                                             |
//+------------------------------------------------------------------+
input bool Accumulative=false;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   int k=100;
   double arr[10];
//--- 创建图表
   CLineChart chart;
//--- 创建图表
   if(!chart.CreateBitmapLabel("SampleHistogrammChart",10,10,600,450))
     {
      Print("Error creating line chart: ",GetLastError());
      return(-1);
     }
   if(Accumulative)
     {
      chart.Accumulative();
      chart.VScaleParams(20*k*10,-10*k*10,20);
     }
   else
      chart.VScaleParams(20*k,-10*k,15);
   chart.ShowScaleTop(false);
   chart.ShowScaleRight(false);
   chart.ShowLegend();
   chart.Filled();
   for(int j=0;j<5;j++)
     {
      for(int i=0;i<10;i++)
        {
         k=-k;
         if(k>0)
            arr[i]=k*(i+10-j);
         else
            arr[i]=k*(i+10-j)/2;
        }
      chart.SeriesAdd(arr,"Item"+IntegerToString(j));
     }
//--- 使用值
   while(!IsStopped())
     {
      int i=rand()%5;
      int j=rand()%10;
      k=rand()%3000-1000;
      chart.ValueUpdate(i,j,k);
      Sleep(200);
     }
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 完成
   chart.Destroy();
   return(0);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Filled

设置指定是否需要填充数据序列定义曲线下的区域的标识。

```
void  Filled(
  const bool  flag=true,  //标识
  )
```

参数

*flag=true*

[in] 标识值：

- true — 填充曲线下的区域
- false —不填充曲线下的区域

© 2000-2025, MetaQuotes Ltd.

# Create

创建图形资源的虚拟方法。

```
virtual bool  Create(
  const string       name,    //名称
  const int          width,   //宽度
  const int          height,  //高度
  ENUM_COLOR_FORMAT  clrfmt,  //格式
  )
```

参数

*名称*

[in] 图形资源名称的基础。资源名称是在添加伪随机字符串创建时生成的。

*宽度*

[in] 以像素为单位的宽度（沿X轴的大小）。

*高度*

[in] 以像素为单位的高度（沿Y轴的大小）。

*clrfmt*

[in] 颜色处理方法。查看ResourceCreate()函数描述来了解更多有关颜色处理方法。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesAdd

添加新的数据序列。

```
bool  SeriesAdd(
  const double&  value[],  // 值
  const string   descr,    // 标签
  const uint     clr,      // 颜色
  )
```

参数

*value[]*

[in] 数据序列。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesInsert

插入数据序列到图表。

```
bool  SeriesInsert(
  const uint     pos,        // 索引编号
  const double&  value[],    //值
  const string   descr,      // 标签
  const uint     clr,        // 颜色
  )
```

参数

*pos*

[in] 插入索引。

*value[]*

[in] 数据序列。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesUpdate

图表上的更新数据序列。

```
bool  SeriesUpdate(
  const uint     pos,        // 索引编号
  const double&  value[],    //值
  const string   descr,      // label
  const uint     clr,        // 颜色
  )
```

参数

*pos*

[in] 序列索引 —从0开始的添加序列号。

*value[]*

[in] 数据序列的新值。

*descr*

[in] 序列标签。

*clr*

[in] 序列显示颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesDelete

从图表删除数据序列。

```
bool  SeriesDelete(
  const uint  pos,  //索引编号
  )
```

参数

*pos*

[in] 序列索引 — 从0开始的添加序列号。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueUpdate

更新指定序列的指定值。

```
bool  ValueUpdate(
  const uint   series,  // 序列索引
  const uint   pos,     // 元素索引
  double       value,   // 值
  )
```

参数

*series*

[in] 序列索引 —从0开始的添加序列号。

*pos*

[in] 序列元素索引。

*value*

[in] 新值。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# DrawChart

绘制曲线及其所有元素的虚拟方法。

```
virtual void  DrawChart()
```

© 2000-2025, MetaQuotes Ltd.

# DrawData

为指定序列绘制曲线的虚拟方法。

```
virtual void  DrawData(
  const uint  index,  // 索引编号
  )
```

参数

*index*

[in] 序列索引 —从0开始的添加序列号。

© 2000-2025, MetaQuotes Ltd.

# CalcArea

计算数据序列定义曲线下的区域。

```
double  CalcArea(
  const uint  index,  // 索引编号
  )
```

参数

*index*

[in] 序列索引 —从0开始的添加序列号。

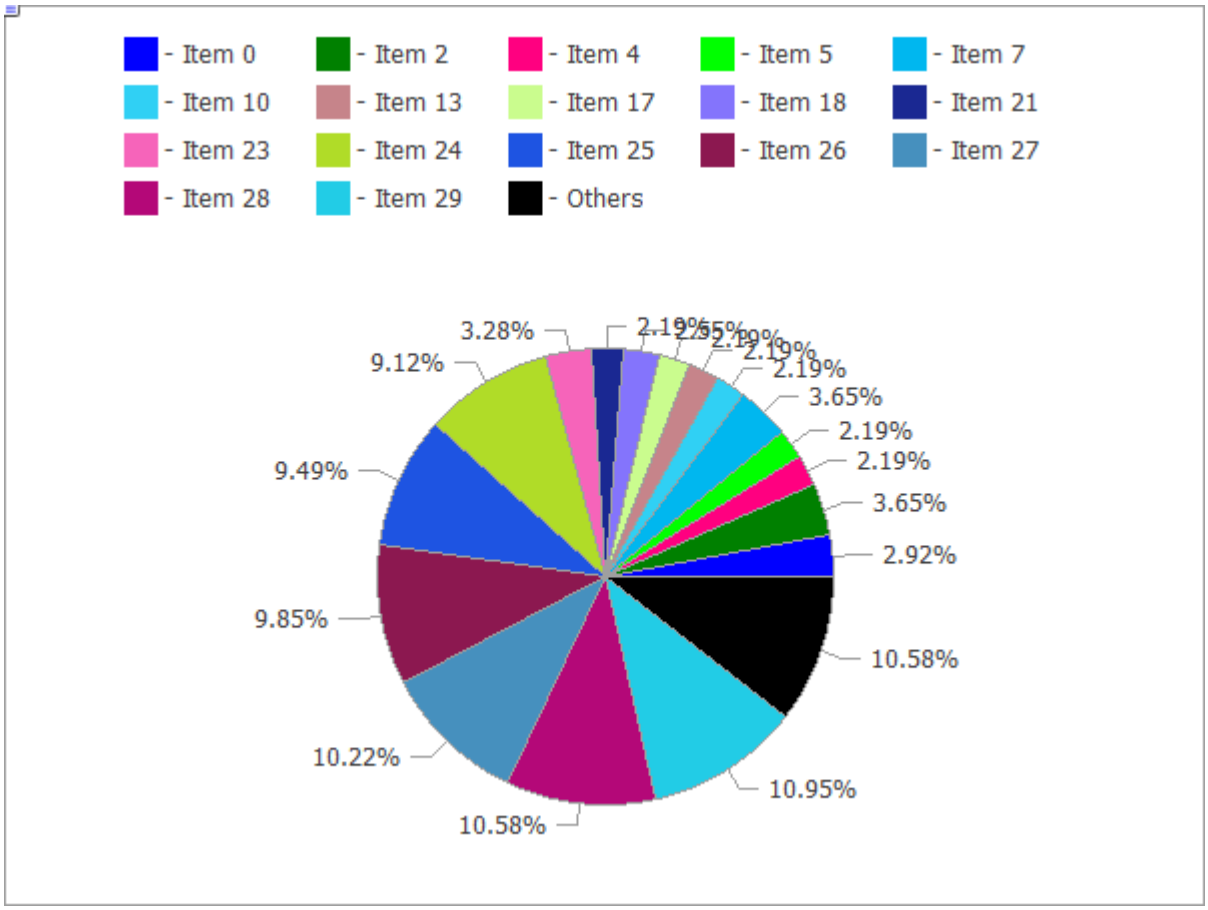
返回值

通过数据序列定义的曲线限定的图形区域。

© 2000-2025, MetaQuotes Ltd.

# CPieChart

绘制饼图的类。




下面提供图形上方的代码。

## 描述

该类中包含的方法专为饼图的全比例操作而设计，从创建图形资源到设计片段标签。

## 声明

```
class CPieChart : public CChartCanvas
```

## 主题

```
#include <Canvas\Charts\PieChart.mqh>
```

## 继承体系

CCanvas

CChartCanvas

CPieChart

## 类函数

© 2000-2025, MetaQuotes Ltd.

| 方法 | 动作 |
| --- | --- |
| Create | 创建图形资源的虚拟方法。 |
| SeriesSet | 设置将在饼图上显示的序列值。 |
| ValueAdd | 添加新值到饼图（到结束）。 |
| ValueInsert | 插入新值到饼图（指定位置）。 |
| ValueUpdate | 更新饼图上的值（指定位置）。 |
| ValueDelete | 从饼图删除值（指定位置）。 |
| DrawChart | 绘制饼图及其全部元素的虚拟方法。 |
| DrawPie | 绘制对应指定值的饼图片段。 |
| LabelMake | 基于其值和原标签生成片段标签。 |

方法继承自类CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

方法继承自类CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags, ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

示例

```
//+------------------------------------------------------------------+
//|                                                PieChartSample.mq5 |
//|                  Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property description "Example of using pie chart"
//---
#include <Canvas\Charts\PieChart.mqh>
//+------------------------------------------------------------------+
//| 输入                                                             |
//+------------------------------------------------------------------+
input int      Width=600;
input int      Height=450;
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
int OnStart(void)
  {
//--- 检查
   if(Width<=0 || Height<=0)
     {
      Print("Too simple.");
      return(-1);
     }
//--- 创建图表
   CPieChart pie_chart;
   if(!pie_chart.CreateBitmapLabel("PieChart",10,10,Width,Height))
     {
      Print("Error creating pie chart: ",GetLastError());
      return(-1);
     }
   pie_chart.ShowPercent();
//--- 绘制
   for(uint i=0;i<30;i++)
     {
      pie_chart.ValueAdd(100*(i+1),"Item "+IntegerToString(i));
      Sleep(10);
     }
   Sleep(2000);
//--- 禁用图例
   pie_chart.LegendAlignment(ALIGNMENT_LEFT);
   Sleep(2000);
//--- 禁用图例
   pie_chart.LegendAlignment(ALIGNMENT_RIGHT);
   Sleep(2000);
//--- 禁用图例
   pie_chart.LegendAlignment(ALIGNMENT_TOP);
   Sleep(2000);
//--- 禁用图例
   pie_chart.ShowLegend(false);
   Sleep(2000);
//--- 禁用百分比
   pie_chart.ShowPercent(false);
   Sleep(2000);
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 禁用描述符
   pie_chart.ShowDescriptors(false);
   Sleep(2000);
//--- 全部启用
   pie_chart.ShowLegend();
   pie_chart.ShowValue();
   pie_chart.ShowDescriptors();
   Sleep(2000);
//--- 或这样
   pie_chart.ShowFlags(FLAG_SHOW_LEGEND|FLAG_SHOW_DESCRIPTORS|FLAG_SHOW_PERCENT);
   uint total=pie_chart.DataTotal();
//--- 使用值
   for(uint i=0;i<total && !IsStopped();i++)
     {
      pie_chart.ValueUpdate(i,100*(rand()%10+1));
      Sleep(1000);
     }
//--- 使用颜色
   for(uint i=0;i<total && !IsStopped();i++)
     {
      pie_chart.ColorUpdate(i%total,RandomRGB());
      Sleep(1000);
     }
//--- 循环
   while(!IsStopped())
     {
      pie_chart.DataOffset(pie_chart.DataOffset()+1);
      Sleep(200);
     }
//--- 完成
   pie_chart.Destroy();
   return(0);
  }
//+------------------------------------------------------------------+
//| 随机RGB颜色                                                      |
//+------------------------------------------------------------------+
uint RandomRGB(void)
  {
   return(XRGB(rand()%255,rand()%255,rand()%255));
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建图形资源的虚拟方法。

```
virtual bool  Create(
  const string        name,    //名称
  const int           width,   //宽度
  const int           height,  //高度
  ENUM_COLOR_FORMAT   clrfmt,  //格式
  )
```

参数

*名称*

[in] 图形资源名称的基础。资源名称是在添加伪随机字符串创建时生成的。

*宽度*

[in] 以像素为单位的宽度（沿X轴的大小）。

*高度*

[in] 以像素为单位的高度（沿Y轴的大小）。

*clrfmt*

[in] 颜色处理方法。查看ResourceCreate()函数描述来了解更多有关颜色处理方法。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# SeriesSet

设置将在饼图上显示的序列值。

```
bool  SeriesSet(
  const double&  value[],  // 值
  const string&  text[],   // 标签
  const uint&    clr[],    // 颜色
  )
```

参数

*value[]*

[in] 值数组。

*text[]*

[in] 值标签数组。

*clr[]*

[in] 值颜色数组。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueAdd

添加新值到饼图 (到结束)。

```
bool  ValueAdd(
  const double  value,  // 值
  const string  descr,  // 标签
  const uint    clr,    // 颜色
  )
```

参数

*value*

[in] 值

*descr*

[in] 值标签。

*clr*

[in] 值颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueInsert

插入新值到饼图（指定位置）。

```
bool  ValueInsert(
  const uint    pos,    // 索引编号
  const double  value,  // 值
  const string  descr,  // 标签
  const uint    clr,    // 颜色
  )
```

参数

*pos*

[in] 插入索引。

*value*

[in] 值

*descr*

[in] 值标签。

*clr*

[in] 值颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueUpdate

更新饼图上的值（指定位置）。

```
bool  ValueUpdate(
  const uint    pos,    // 索引编号
  const double  value,  // 值
  const string  descr,  // 标签
  const uint    clr,    // 颜色
  )
```

参数

*pos*

[in] 值的索引—从0开始的添加序列号。

*value*

[in] 值

*descr*

[in] 值标签。

*clr*

[in] 值颜色。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# ValueDelete

从饼图删除值（指定位置）。

```
bool  ValueDelete(
  const uint  pos,  //索引编号
  )
```

参数

*pos*

[in] 值的索引—从0开始的添加序列号。

返回值

如果成功返回true，否则—false。

© 2000-2025, MetaQuotes Ltd.

# DrawChart

绘制饼图及其全部元素的虚拟方法。

```
virtual void  DrawChart()
```

© 2000-2025, MetaQuotes Ltd.

# DrawPie

绘制对应指定值的饼图片段。

```
void  DrawPie(
  double       fi3,    // 饼图中心发出的射线角，定义第一个弧形边框
  double       fi4,    // 饼图中心发出的射线角，定义第二个弧形边框
  int          idx,    // 索引编号
  CPoint&      p[],   //
  const uint   clr,    //
  )
```

参数

*fi3*

[in] 弧度角，定义第一个弧形边框。

*fi4*

[in] 弧度角，定义第二个弧形边框。

*idx*

[in] 片段对应的值索引。

*p[]*

[in] 用于绘制片段的参考点数组(x, y)。

*clr*

[in] 片段颜色。

© 2000-2025, MetaQuotes Ltd.

# LabelMake

基于其值和原标签生成片段标签。

```
string  LabelMake(
  const string  text,     // 标签
  const double  value,    // 值
  const bool    to_left,  // 标识
  )
```

参数

*text*

[in] 标签。

*value*

[in] 值。

*to_left*

[in] 定义标签布局顺序：

- true —标签，然后值。
- false —值，然后标签。

返回值

片段标签。

© 2000-2025, MetaQuotes Ltd.

# 3D图形

这部分介绍的是用于三维图形开发的类。这些类基于使用DirectX的函数。CCnavas3D是一个基类，包含管理相机和光线的方法，以及图形资源的管理器 -纹理、着色器、顶点缓冲区、索引和着色器参数。

若要开始使用该函数库，只需阅读文章如何在MetaTrader5中使用DirectX创建3D图形。

此外，还有场景基对象的类，比如盒子，用户数据上的三维表面和任意网格。

© 2000-2025, MetaQuotes Ltd.

# CCanvas3D

CCanvas3D是一个用于在图表上简化3D对象创建和可视化的类。

## 描述

CCanvas3D以动态3D图形的形式大幅简化了大量数据的创建和可视化。CCanvas3D类包含管理相机和光线的方法，以及提供了用于创建图形资源的资源管理器 - 纹理、着色器、顶点缓冲区、索引和着色器参数。

此外，程序库还包括场景基对象的类，比如盒子，用户数据上的三维表面或任意网格。

显卡应支持DX 11和Shader Model5.0，以便使函数正常工作。

## 声明

```
class CCanvas
```

## 标题

```
#include <Canvas\Canvas.mqh>
```

## 继承架构

CCanvas

CCanvas3D

## 按组划分类函数

| 创建/删除 | 描述 |
| --- | --- |
| 创建 | 创建用于渲染3D场景而无需绑定到图表对象的图形资源。 |
| Attach | 从OBJ_BITMAP_LABEL对象获得图形资源并将其附加到CCanvas类的实例。 |
| ObjectAdd | 将对象添加到3D场景以进行后续渲染。 |
| Destroy | 破坏图形资源并发布图形3D环境。 |
| 光线 | |
| AmbientColorSet | 设置环境全方位光线的颜色和强度。 |
| AmbientColorGet | 获取环境全方位光线的颜色和强度。 |
| LightDirectionSet | 设置定向光源的方向。 |
| LightDirectionGet | 获取定向光源的方向。 |
| LightColorSet | 设置定向光源的颜色和强度。 |
| LightColorGet | 获取定向光源的颜色和强度。 |
| 相机 | |

© 2000-2025, MetaQuotes Ltd.

| 创建/删除 | 描述 |
| --- | --- |
| ProjectionMatrixSet | 计算并设置3D坐标投影矩阵到2D帧。 |
| ProjectionMatrixGet | 获取3D坐标投影矩阵到2D帧。 |
| ViewMatrixSet | 设置3D场景视图矩阵。 |
| ViewMatrixGet | 返回3D场景视图矩阵。 |
| ViewPositionSet | 设置3D场景的视点。 |
| ViewRotationSet | 设置在3D场景中注视的方向。 |
| ViewTargetSet | 设置注视点的坐标。 |
| ViewUpDirectionSet | 在3D空间中设置上边框的方向。 |
| 渲染 | |
| Render | 渲染画帧内部缓冲区中的所有场景对象，以供后续显示。 |
| RenderBegin | 准备用于渲染新画帧的图形环境。 |
| RenderEnd | 将已渲染的画帧复制到内部缓冲区，并在必要时更新图表图像。 |
| 获取资源 | |
| DXContext | 获取图形环境句柄。 |
| DXDispatcher | 获取资源分配器句柄。 |
| InputScene | 获取指向场景参数缓冲区的指针。 |

© 2000-2025, MetaQuotes Ltd.

# AmbientColorGet

获取环境全方位光线的颜色和强度。

```
void  AmbientColorGet(
   DXColor  &ambient_color      // 全方位光线的颜色和强度
   );
```

参数

*&ambient_color*

[out] 全方位光线的颜色。

返回值

否

注意

强度存储在DXColor结构的alpha通道中。

© 2000-2025, MetaQuotes Ltd.

# AmbientColorSet

设置环境全方位光线的颜色和强度。

```
void  AmbientColorSet(
   const DXColor  &ambient_color      // 全方位光线的颜色和强度
   );
```

参数

*&ambient_color*

[in] 全方位光线的颜色。

返回值

否

注意

强度在DXColor结构的alpha通道中设置。

© 2000-2025, MetaQuotes Ltd.

# Attach

从OBJ_BITMAP_LABEL对象获得图形资源并将其附加到CCanvas类的实例。

```
bool  Attach(
   const long          chart_id,                          // 图表ID
   const string        objname,                           // 对象名称
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // 颜色处理方法
   )
```

为OBJ_BITMAP_LABEL对象创建一个图形资源并将其附加到CCanvas类的实例。

```
bool  Attach(
   const long          chart_id,                          // 图表ID
   const string        objname,                           // 对象名称
   const int           width,                             // 以像素表示的图像宽度
   const int           height,                            // 以像素表示的图像高度
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // 颜色处理方法
   )
```

参数

*chart_id*

[in] 图表 ID。

*objname*

[in] 图形对象的名称。

*width*

[in] 资源中的画帧宽度。

*height*

[in] 画帧高度。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

[in] 颜色处理方法。参看ResourceCreate()函数描述来了解更多有关颜色处理方法。

注意

true -如果成功，false - 如果添加图形对象失败。

© 2000-2025, MetaQuotes Ltd.

# 创建

创建用于渲染3D场景而无需绑定到图表对象的图形资源。

```
virtual bool  Create(
   const string       name,                                  // 图形对象名称
   const int          width,                                 // 宽度
   const int          height,                                // 高度
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // 颜色格式
   );
```

参数

*name*

[in] 图形对象名称。

*width*

[in] 画帧宽度。

*height*

[in] 画帧高度。

*clrfmt=COLOR_FORMAT_XRGB_NOALPHA*

[in] 颜色处理方法。参看ResourceCreate()函数描述来了解更多有关颜色处理方法。

注意

true - 如果创建资源，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Destroy

破坏图形资源并发布图形3D环境。

```
virtual void  Destroy()
```

返回值

否

注意

如果图形资源已绑定到图表对象，则删除后者。

© 2000-2025, MetaQuotes Ltd.

# DXContext

获取图形环境句柄。

```
int  DXContext()
```

返回值

图表环境句柄。

© 2000-2025, MetaQuotes Ltd.

# DXDispatcher

获取资源分配器句柄。

```
CDXDispatcher*  DXDispatcher()
```

返回值

资源分配器句柄。

© 2000-2025, MetaQuotes Ltd.

# InputScene

获取指向场景参数缓冲区的指针。

```
CDXInput*  InputScene()
```

返回值

指向场景参数缓冲区的指针。

© 2000-2025, MetaQuotes Ltd.

# LightColorGet

获取定向光源的颜色和强度。

```
void  LightColorGet(
   DXColor  &light_color      // 定向光线颜色和强度
   );
```

参数

*&light_color*

[out] 定向光线颜色和强度。

返回值

无。

注意

强度存储在DXColor结构的alpha通道中。

© 2000-2025, MetaQuotes Ltd.

# LightColorSet

设置定向光源的颜色和强度。

```
void  LightColorSet(
   const DXColor  &light_color      // 定向光线颜色和强度
   );
```

参数

*&light_color*

[in] 定向光线颜色和强度。

返回值

无。

注意

强度在DXColor结构的alpha通道中设置。

© 2000-2025, MetaQuotes Ltd.

# LightDirectionGet

获取定向光源的方向。

```
void  LightDirectionGet(
   DXVector3  &light_direction      // 方向矢量
   );
```

参数

*&light_direction*

[out] 方向矢量。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# LightDirectionSet

设置定向光源的方向。

```
void  LightDirectionSet(
   const DXVector3  &light_direction      // 方向矢量
   );
```

参数

*&light_direction*

[in] 方向矢量。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ObjectAdd

将对象添加到3D场景以进行后续渲染。

```
bool  ObjectAdd(
   CDXObject  *object      // 指向对象的指针
   );
```

参数

*object*

[in] 指向从CDXObject抽象类派生的类的实例的指针。

返回值

true -如果成功，false - 如果添加3D图形对象失败。

© 2000-2025, MetaQuotes Ltd.

# ProjectionMatrixGet

获取3D坐标投影矩阵到2D帧。

```
void  ProjectionMatrixGet(
   DXMatrix  &projection_matrix      // 投影矩阵
   );
```

参数

*&projection_matrix*

[out] 投影矩阵。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ProjectionMatrixSet

计算并设置3D坐标投影矩阵到2D帧。

```
void  ProjectionMatrixSet(
   float  fov,              // 视角
   float  aspect_ratio,     // 画帧长宽比
   float  z_near,           //
   float  z_far             //
   );
```

参数

*fov*

[in] 视角宽度（以弧度表示）以创建场景投影。

*aspect_ratio*

[in] 2D画帧长宽比。

*z_near*

[in] 到近裁剪平面的距离。

*z_far*

[in] 到远裁剪平面的距离。

返回值

无。

注意

2D画帧只显示落入指定视角且位于近裁剪平面和远裁剪平面之间的3D对象的投影。

© 2000-2025, MetaQuotes Ltd.

# Render

渲染画帧内部缓冲区中的所有场景对象，以供后续显示。

```
bool  Render(
   uint  flags,                  // 标识组合
   uint  background_color=0      // 背景颜色
   );
```

参数

*flags*

[in] 设置渲染模式的标识组合。可能值：
DX_CLEAR_COLOR -使用*background_color*清除图像缓冲区。
DX_CLEAR_DEPTH -清除深度缓冲区。

*background_color=0*

[in] 3D场景背景颜色。

返回值

true -如果成功，false -如果渲染失败。

注意

调用Render()不会更新图表上的场景。相反，它只更新图像的内部缓冲区。应该明确调用Update()方法来呈现更新的画帧。

Render()包括RenderBegin和RenderEnd()调用。

© 2000-2025, MetaQuotes Ltd.

# RenderBegin

准备用于渲染新画帧的图形环境。

```
virtual bool  RenderBegin(
   uint  flags,                  // 标识组合
   uint  background_color=0      // 背景颜色
   );
```

参数

*flags*

[in] 设置渲染模式的标识组合。可能值：
DX_CLEAR_COLOR -使用*background_color*清除图像缓冲区。
DX_CLEAR_DEPTH -清除深度缓冲区。

*background_color=0*

[in] 3D场景背景颜色。

返回值

true -如果成功，false - 如果更新着色器失败。

© 2000-2025, MetaQuotes Ltd.

# RenderEnd

将已渲染的画帧复制到内部缓冲区，并在必要时更新图表图像。

```
virtual bool  RenderEnd(
   bool  redraw=false      // 更新标识
   );
```

参数

*redraw=false*

[in] 图表重绘必要性的标识。

返回值

true - 成功，否则 — false。

© 2000-2025, MetaQuotes Ltd.

# ViewMatrixGet

返回3D场景视图矩阵。

```
void  ViewMatrixGet(
   DXMatrix  &view_matrix      // 视图矩阵
   );
```

参数

*&view_matrix*

[out] 相机在3D空间的位置和方向的视图矩阵设置。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ViewMatrixSet

设置3D场景视图矩阵。

```
void  ViewMatrixSet(
   const DXMatrix  &view_matrix      // 视图矩阵
   );
```

参数

*&view_matrix*

[in] 相机在3D空间的位置和方向的视图矩阵设置。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ViewPositionSet

设置3D场景的视点。

```
void  ViewPositionSet(
   const DXVector3  &position      // 视点位置
   );
```

参数

*&position*

[in] 设置3D场景中的视点位置。

返回值

无。

注意

使用ViewPositionSet()设置视点位置会更改在ViewMatrixGet()中获得的视图矩阵。

© 2000-2025, MetaQuotes Ltd.

# ViewRotationSet

设置在3D场景中注视的方向。

```
void  ViewRotationSet(
   const DXVector3  &rotation      // 转角矢量
   );
```

参数

*&rotation*

[in] 矢量设置Euler角度，以计算在3D场景中注视的方向。

返回值

无。

注意

使用ViewRotationSet()设置注视方向会更改在ViewMatrixGet()中获得的视图矩阵。

© 2000-2025, MetaQuotes Ltd.

# ViewTargetSet

设置注视点的坐标。

```
void  ViewTargetSet(
   const DXVector3  &target      // 目标坐标
   );
```

参数

*&target*

[in] 注视点的坐标。

返回值

无。

注意

用于在移动视点时将视线固定在一个场景点上。

使用ViewRotationSet()设置新目标坐标会更改在ViewMatrixGet()中获得的视图矩阵。

ViewTargetSet()与ViewUpDirectionSet()一起使用，来定义注视方向。

© 2000-2025, MetaQuotes Ltd.

# ViewUpDirectionSet

在3D空间中设置上边框的方向。

```
void  ViewUpDirectionSet(
   const DXVector3  &up_direction      // 向上的方向
   );
```

参数

*&up_direction*

[in] 在3D空间中上部边框的方向。

返回值

无。

注意

使用ViewUpDirectionSet()设置新方向会更改在ViewMatrixGet()中获得的视图矩阵。

ViewUpDirectionSet()与ViewTargetSet()一起使用，来定义注视方向。

© 2000-2025, MetaQuotes Ltd.

# CChart

CChart 类用于简便地访问 "Chart" 图形对象属性。

## 描述

CChart 类可供访问 "Chart" 对象属性。

## 声明

```
class CChart : public CObject
```

## 标称库文件

```
#include <Charts\Chart.mqh>
```

## 继承体系

CObject
  CChart

## 类方法

| 访问保护数据 | |
|---|---|
| ChartID | 获取图表标识符 |
| 常用属性 | |
| Mode | 获取/设置 "模式" 属性的值（柱线，蜡烛或线条） |
| Foreground | 获取/设置 "Foreground" 属性的值（前景色） |
| 平移 | 获取/设置 "Shift" 属性的值（平移） |
| ShiftSize | 获取/设置 "ShiftSize" 属性的值（百分比） |
| AutoScroll | 获取/设置 "AutoScroll" 属性的值（自动滚动） |
| Scale | 获取/设置 "Scale" 属性的值（缩放） |
| ScaleFix | 获取/设置 "ScaleFix" 属性的值（固定或非固定图表缩放） |
| ScaleFix_11 | 获取/设置 "ScaleFix_11" 属性的值（图表缩放是否为 1:1） |
| FixedMax | 获取/设置 "FixedMax" 属性的值（固定最大价格） |
| FixedMin | 获取/设置 "FixedMin" 属性的值（固定最小价格） |
| ScalePPB | 获取/设置 "ScalePPB" 属性的值（缩放是否为 "每根柱线的点数"） |
| PointsPerBar | 获取/设置 "PointsPerBar" 属性的值（每根柱线的点数） |
| 显示属性 | |
| ShowOHLC | 获取/设置 "ShowOHLC" 属性的值（显示开高低收价） |

© 2000-2025, MetaQuotes Ltd.

| 访问保护数据 | |
|---|---|
| ShowLineBid | 获取/设置"ShowLineBid"属性的值（显示卖出价位线） |
| ShowLineAsk | 获取/设置"ShowLineAsk"属性的值（显示买入价位线） |
| ShowLastLine | 获取/设置"ShowLastLine"属性的值（显示最后交易价位线） |
| ShowPeriodSep | 获取/设置"ShowPeriodSep"属性的值（显示周期分割线） |
| ShowGrid | 获取/设置"ShowGrid"属性的值（显示网格） |
| ShowVolumes | 获取/设置"ShowVolumes"属性的值（已开仓位的交易量和价位颜色） |
| ShowObjectDescr | 获取/设置"ShowObjectDescr"属性的值（显示图形对象的描述） |
| ShowDateScale | 设置"ShowDateScale"属性的值（图表的日期缩放） |
| ShowPriceScale | 设置"ShowPriceScale"属性的值（图表的价格缩放） |
| 颜色属性 | |
| ColorBackground | 获取/设置"ColorBackground"属性的值（图表的背景颜色） |
| ColorForeground | 获取/设置"ColorForeground"属性的值（图表的轴线，缩放和OHLC字符串颜色） |
| ColorGrid | 获取/设置"ColorGrid"属性的值（网格颜色） |
| ColorBarUp | 获取/设置"ColorBarUp"属性的值（阳柱线及其影线和蜡烛实体轮廓的颜色） |
| ColorBarDown | 获取/设置"ColorBarDown"属性的值（阴柱线及其影线和蜡烛实体轮廓的颜色） |
| ColorCandleBull | 获取/设置"ColorCandleBull"属性的值（阳线实体颜色） |
| ColorCandleBear | 获取/设置"ColorCandleBear"属性的值（阴线实体颜色） |
| ColorChartLine | 获取/设置"ColorChartLine"属性的值（图表线和十字星蜡烛颜色） |
| ColorVolumes | 获取/设置"ColorVolumes"属性的值（已开仓位的交易量和价位颜色） |
| ColorLineBid | 获取/设置"ColorLineBid"属性的值（卖价线颜色） |
| ColorLineAsk | 获取/设置"ColorLineAsk"属性的值（买价线颜色） |
| ColorLineLast | 获取/设置"ColorLineLast"属性的值（最后一笔成交价位颜色） |
| ColorStopLevels | 获取/设置"ColorStopLevels"属性的值（止损和止盈位的颜色） |
| 只读属性 | |

© 2000-2025, MetaQuotes Ltd.

| 访问保护数据 | |
|---|---|
| VisibleBars | 获取图表可视柱线总数 |
| WindowsTotal | 获取图表窗口总数，包括图表指标的子窗口 |
| WindowIsVisible | 获取指定图表窗口的可视性标志 |
| WindowHandle | 获取图表的窗口句柄 (HWND) |
| FirstVisibleBar | 获取图表的首根可视柱线号码 |
| WidthInBars | 获取窗口宽度，单位为柱线。 |
| WidthInPixels | 获取子窗口宽度，单位为像素。 |
| HeightInPixels | 获取子窗口高度，单位为像素。 |
| PriceMin | 获取指定子窗口的最小价格 |
| PriceMax | 获取指定子窗口的最大价格 |
| 属性 | |
| Attach | 将当前图表分配到类实例 |
| FirstChart | 将客户端的首个图表分配到类实例 |
| NextChart | 将客户端的下一个图表分配到类实例 |
| Open | 以指定参数打开图表并将其分配到类实例 |
| Detach | 从类实例中分离图表 |
| Close | 关闭已分配到类实例的图表 |
| BringToTop | 在其它图表之上显示图表 |
| EventObjectCreate | 设置在图表上创建新对象事件的发送通知标志 |
| EventObjectDelete | 设置在图表上删除对象事件的发送通知标志 |
| 指标 | |
| IndicatorAdd | 添加指定句柄的指标至指定的图表子窗口 |
| IndicatorDelete | 从指定的图表子窗口删除指定名称的指标 |
| IndicatorsTotal | 返回应用于指定图表子窗口中的所有指标数量 |
| IndicatorName | 返回指定图表子窗口中指标的简称 |
| 遍历 | |
| Navigate | 遍历图表 |
| 访问 MQL5 API | |
| Symbol | 获得图表品名 |
| Period | 获得图表周期 |

© 2000-2025, MetaQuotes Ltd.

| 访问保护数据 | |
|---|---|
| 重绘 | 重绘已分配到类实例的图表 |
| GetInteger | 此函数返回相应的对象属性值 |
| SetInteger | 设置整数类型属性新数值 |
| GetDouble | 此函数返回相应的对象属性值 |
| SetDouble | 设置双精度类型属性新数值 |
| GetString | 此函数返回相应的对象属性值 |
| SetString | 设置字符类型属性新值 |
| SetSymbolPeriod | 改变已分配到类实例的图表的品名和周期 |
| ApplyTemplate | 应用指定模板至图表 |
| ScreenShot | 创建指定图表的截屏并将之保存为 .gif 文件 |
| WindowOnDropped | 获取与对象（智能程序或脚本）落点相应的图表子窗口号码 |
| PriceOnDropped | 获取与对象（智能程序或脚本）落点相应的价格坐标 |
| TimeOnDropped | 获取与对象（智能程序或脚本）落点相应的时间坐标 |
| XOnDropped | 获取与对象（智能程序或脚本）落点相应的 X 坐标 |
| YOnDropped | 获取与对象（智能程序或脚本）落点相应的 Y 坐标 |
| 输入/输出 | |
| virtual Save | 保存对象参数至文件 |
| virtual Load | 从文件加载对象参数 |
| virtual Type | 获取图形对象类型标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

© 2000-2025, MetaQuotes Ltd.

# ChartID

返回图表标识符。

```
long  ChartID() const
```

返回值

已分配到类实例中的图表标识符。如果无分配对象，则返回 -1。

© 2000-2025, MetaQuotes Ltd.

## Mode (Get方法)

获取 "Mode" 属性的值 (柱线, 蜡烛或线条)。

```
ENUM_CHART_MODE  Mode() const
```

返回值

已分配到类实例中的对象 "Mode" 属性值。如果无分配对象, 则返回 WRONG_VALUE。

## Mode (Set方法)

设置 "Mode" 属性的值 (柱线, 蜡烛或线条)。

```
bool  Mode(
   ENUM_CHART_MODE  mode       // 新图表模式
   )
```

参数

*mode*

[输入] 图表模式 (柱线, 蜡烛或线条) ENUM_CHART_MODE 枚举。

返回值

true 如果成功, false 如果模式未改变。

© 2000-2025, MetaQuotes Ltd.

## Foreground (Get方法)

获取 "Foreground" 属性的值。

```
bool  Foreground() const
```

返回值

已分配到类实例中的对象 "Foreground" 属性值。如果无分配对象，则返回 false。

## Foreground (Set方法)

设置 "Foreground" 属性的新值。

```
bool  Foreground(
   bool  foreground      // 新标志值
   )
```

参数

*foreground*

[输入] "Foreground" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Shift (Get 方法)

获取 "Shift" 属性的值。

```
bool  Shift() const
```

返回值

已分配到类实例中的对象 "Shift" 属性值。如果无分配对象，则返回 false。

## Shift (Set 方法)

设置 "Shift" 属性的新值。

```
bool  Shift(
   bool  shift       // 新标志值
   )
```

参数

*shift*

[输入] "Shift" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShiftSize (Get方法)

获取"ShiftSize"属性的值(百分比)

```
double  ShiftSize() const
```

返回值

已分配到类实例中的对象"ShiftSize"属性值。如果无分配对象,则返回 EMPTY_VALUE。

## ShiftSize (Set方法)

设置"平移"属性的新值(百分比)。

```
bool  ShiftSize(
   double  shift_size      // 新属性值
   )
```

参数

*shift_size*

[输入]"ShiftSize"属性的新值(百分比)。

返回值

true 如果成功,false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## AutoScroll (Get 方法)

获取 "AutoScroll" 属性的值。

```
bool  AutoScroll() const
```

返回值

已分配到类实例中的对象 "AutoScroll" 属性值。如果无分配对象，则返回 false。

## AutoScroll (Set 方法)

设置 "AutoScroll" 属性的值。

```
bool  AutoScroll(
   bool  autoscroll      // 新标志值
   )
```

参数

*autoscroll*

[输入] "Autoscroll" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## Scale (Get方法)

获取"Scale"属性的值。

```
int  Scale() const
```

返回值

已分配到类实例中的对象"Scale"属性值。如果无分配对象，则返回0。

## Scale (Set方法)

设置"Scale"属性的新值。

```
bool  Scale(
   int  scale      // 新值
   )
```

参数

*scale*

[输入]"Scale"属性的新值。

返回值

true如果成功，false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ScaleFix (Get方法)

获取/设置"ScaleFix"属性的值 (固定或非固定图表缩放)。

```
bool  ScaleFix() const
```

返回值

已分配到类实例中的对象"ScaleFix"属性值。如果无分配对象，则返回 false。

## ScaleFix (Set方法)

设置"ScaleFix"属性的新值。

```
bool  ScaleFix(
   bool  scale_fix      // 新值
   )
```

参数

*scale_fix*

[输入]"ScaleFix"属性新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ScaleFix_11 (Get方法)

获取"ScaleFix_11"属性的值 (图表缩放是否为 1:1)。

```
bool  ScaleFix_11() const
```

返回值

已分配到类实例中的对象"ScaleFix_11"属性值。如果无分配对象，则返回 false。

## ScaleFix_11 (Set方法)

设置"ScaleFix_11"属性的新值。

```
bool  ScaleFix_11(
   string  scale_11      // 新值
   )
```

参数

*scale_11*

[输入]"ScaleFix_11"属性新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## FixedMax (Get方法)

获取"FixedMax"属性的值 (固定最大价格)。

```
double  FixedMax() const
```

返回值

已分配到类实例中的对象"FixedMax"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## FixedMax (Set方法)

设置"FixedMax"属性的新值。

```
bool  FixedMax(
   double  max       // 新的固定最大值
   )
```

参数

*max*

[输入]"FixedMax"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## FixedMin (Get方法)

获取"FixedMin"属性的值 (固定最小价格)。

```
double  FixedMin() const
```

返回值

已分配到类实例中的对象"FixedMin"属性值。如果无分配对象，则返回 EMPTY_VALUE。

## FixedMin (Set方法)

设置"FixedMin"属性的新值。

```
bool  FixedMax(
   double  min      // 新的固定最小值
   )
```

参数

*max*

[输入]"FixedMin"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## PointsPerBar (Get方法)

获取"PointsPerBar"属性的值(每根柱线的点数)。

```
double  PointsPerBar() const
```

返回值

已分配到类实例中的对象"PointsPerBar"属性值。如果无分配对象,则返回 EMPTY_VALUE。

## PointsPerBar (Set方法)

设置"PointsPerBar"属性的新值。

```
bool  PointsPerBar(
   double  ppb       // 新缩放(每根柱线的点数)
   )
```

参数

*ppb*

[输入]新缩放值(每根柱线的点数)。

返回值

true 如果成功,false 如果缩放不可改变。

© 2000-2025, MetaQuotes Ltd.

## ScalePPB (Get方法)

获取"ScalePPB"属性的值(缩放是否为"point per bar")。

```
bool  ScalePPB() const
```

返回值

已分配到类实例中的对象"ScalePPB"属性值。如果无分配对象,则返回 false。

## ScalePPB (Set方法)

设置"ScalePPB"属性的新值。

```
bool  ScalePPB(
   bool  scale_ppb      // 新标志值
   )
```

参数

*scale_ppb*

[输入]"ScalePPB"属性的新值。

返回值

true 如果成功,false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowOHLC (Get方法)

获取 "ShowOHLC" 属性的值。

```
bool  ShowOHLC() const
```

返回值

已分配到类实例中的对象 "ShowOHLC" 属性值。如果无分配对象，则返回 false。

## ShowOHLC (Set方法)

设置 "ShowOHLC" 属性的新值。

```
bool  ShowOHLC(
   bool  show      // 新值
   )
```

参数

*show*

[输入] "ShowOHLC" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowLineBid (Get方法)

获取 "ShowLineBid" 属性的值。

```
bool  ShowLineBid() const
```

返回值

已分配到类实例中的对象 "ShowLineBid" 属性值。如果没有已分配图表，则返回 false。

## ShowLineBid (Set方法)

设置 "ShowLineBid" 属性的新值。

```
bool  ShowLineBid(
   bool  show      // 新值
   )
```

参数

*show*

[输入] "ShowLineBid" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowLineAsk (Get方法)

获取"ShowLineAsk"属性的值。

```
bool  ShowLineAsk() const
```

返回值

已分配到类实例中的对象"ShowLineAsk"属性值。如果没有已分配图表，则返回 false。

## ShowLineAsk (Set方法)

设置"ShowLineAsk"属性的新值。

```
bool  ShowLineAsk(
   bool  show      // 新值
   )
```

参数

*show*

[输入]"ShowLineAsk"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowLastLine (Get方法)

获取"ShowLastLine"属性的值。

```
bool  ShowLastLine() const
```

返回值

已分配到类实例中的对象"ShowLastLine"属性值。如果没有已分配图表，则返回 false。

## ShowLastLine (Set方法)

设置"ShowLastLine"属性的新值。

```
bool  ShowLastLine(
   bool  show      // 新标志值
   )
```

参数

*show*

[输入]"ShowLastLine"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowPeriodSep (Get方法)

获取"ShowPeriodSep"属性的值(显示单独周期)。

```
bool  ShowPeriodSep() const
```

返回值

已分配到类实例中的对象"ShowPeriodSep"属性值。如果没有已分配图表,则返回false。

## ShowPeriodSep (Set方法)

设置"ShowPeriodSep"属性的新值。

```
bool  ShowPeriodSep(
   bool  show      // 新值
   )
```

参数

*show*

[输入]"ShowPeriodSep"属性的新值。

返回值

true如果成功,false如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowGrid (Get方法)

获取 "ShowGrid" 属性的值。

```
bool  ShowGrid() const
```

返回值

已分配到类实例中的对象 "ShowGrid" 属性值。如果没有已分配图表，则返回 false。

## ShowGrid (Set方法)

设置 "ShowGrid" 属性的新值。

```
bool  ShowGrid(
   bool  show      // 新值
   )
```

参数

*show*

[输入] "ShowGrid" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowVolumes (Get方法)

获取"ShowVolumes"属性的值。

```
bool  ShowVolumes() const
```

返回值

已分配到类实例中的对象"ShowVolumes"属性值。如果没有已分配图表，则返回 false。

## ShowVolumes (Set方法)

设置"ShowVolumes"属性的新值。

```
bool  ShowVolumes(
   bool  show      // 新值
   )
```

参数

*show*

[输入]"ShowVolumes"属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ShowObjectDescr (Get方法)

获取"ShowObjectDescr"属性的值(显示图形对象的描述)。

```
bool  ShowObjectDescr() const
```

返回值

已分配到类实例中的对象"ShowObjectDescr"属性值。如果没有已分配图表,则返回 false。

## ShowObjectDescr (Set方法)

设置"ShowObjectDescr"属性的新值。

```
bool  ShowObjectDescr(
   bool  show      // 新值
   )
```

参数

*show*

[输入]"ShowObjectDescr"属性的新值。

返回值

true 如果成功,false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# ShowDateScale

设置 "ShowDateScale" 属性的新值。

```
bool  ShowDateScale(
   bool  show      // 新值
   )
```

参数

*show*

[输入] "ShowDateScale" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# ShowPriceScale

设置 "ShowPriceScale" 属性的新值。

```
bool  ShowPriceScale(
   bool  show      // 新值
   )
```

参数

*show*

[输入] "ShowPriceScale" 属性的新值。

返回值

true 如果成功，false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorBackground (Get方法)

获取"ColorBackground"属性的值(图表的背景颜色)。

```
color  ColorBackground() const
```

返回值

已分配到类实例中的对象"ColorBackground"属性值。如果没有已分配图表,则返回CLR_NONE。

## ColorBackground (Set方法)

设置"ColorBackground"属性的新值。

```
bool  ColorBackground(
   color  new_color      // 新背景颜色
   )
```

参数

*new_color*

[输入]新背景颜色。

返回值

true 如果成功, false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorForeground (Get方法)

获取"ColorForeground"属性的值 (图表的轴线, 缩放和 OHLC 字符串颜色)。

```
color  ColorForeground() const
```

返回值

已分配到类实例中的对象 "ColorForeground" 属性值。如果没有已分配图表, 则返回 CLR_NONE。

## ColorForeground (Set方法)

设置"ColorForeground"属性的新值 (轴线, 缩放和 OHLC 字符串)。

```
bool  ColorForeground(
   color  new_color      // 新颜色
   )
```

参数

*new_color*

[输入] 轴线, 缩放和 OHLC 字符串的新颜色。

返回值

true 如果成功, false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorGrid (Get方法)

获取"ColorGrid"属性的值 (网格颜色)。

```
color  ColorGrid() const
```

返回值

已分配到类实例中的对象"ColorGrid"属性值。如果没有已分配图表，则返回 CLR_NONE。

## ColorGrid (Set方法)

设置"ColorGrid"属性的新值。

```
bool  ColorGrid(
   color  new_color      // 新网格颜色
   )
```

参数

*new_color*

[输入] 新网格颜色。

返回值

true 如果成功，false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorBarUp (Get方法)

获取"ColorBarUp"属性的值(阳柱线及其影线和蜡烛实体轮廓的颜色)。

```
color  ColorBarUp() const
```

返回值

已分配到类实例中的对象"ColorBarUp"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorBarUp (Set方法)

设置"ColorBarUp"属性的新值。

```
bool  ColorBarUp(
   color  new_color      // 新阳柱线颜色
   )
```

参数

*new_color*

[输入]新阳柱线颜色。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorBarDown (Get方法)

获取"ColorBarDown"属性的值(阴柱线及其影线和蜡烛实体轮廓的颜色)。

```
color  ColorBarDown() const
```

返回值

已分配到类实例中的对象"ColorBarDown"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorBarDown (Set方法)

设置"ColorBarDown"属性的新值。

```
bool  ColorBarDown(
   color  new_color      // 新阴柱线颜色
   )
```

参数

*new_color*

[输入]新阴柱线颜色。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorCandleBull (Get 方法)

获取 "ColorCandleBull" 属性的值 (阳线实体颜色)。

```
color  ColorCandleBull() const
```

返回值

已分配到类实例中的对象 "ColorCandleBull" 属性值。如果没有已分配图表，则返回 CLR_NONE。

## ColorCandleBull (Set 方法)

设置 "ColorCandleBull" 属性的新值。

```
bool  ColorCandleBull(
   color  new_color      // 新阳线实体颜色
   )
```

参数

*new_color*

[输入] 新阳线实体颜色。

返回值

true 如果成功，false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorCandleBear (Get方法)

获取"ColorCandleBear"属性的值 (阴线实体颜色。)

```
color  ColorCandleBear() const
```

返回值

已分配到类实例中的对象 "ColorCandleBear" 属性值。如果没有已分配图表, 则返回 CLR_NONE。

## ColorCandleBear (Set方法)

设置"ColorCandleBear"属性的新值。

```
bool  ColorCandleBear(
   color  new_color      // 新阴线实体颜色
   )
```

参数

*new_color*
[输入] 新阴线实体颜色。

返回值

true 如果成功, false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorChartLine (Get方法)

获取"ColorChartLine"属性的值(图表线和十字星蜡烛颜色)。

```
color  ColorChartLine() const
```

返回值

已分配到类实例中的对象"ColorChartLine"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorChartLine (Set方法)

设置"ColorChartLine"属性的新值。

```
bool  ColorChartLine(
   color  new_color      // 新图表线颜色
   )
```

参数

*new_color*

[输入]新图表线颜色(十字星蜡烛)。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorVolumes (Get方法)

获取"ColorVolumes"属性的值 (已开仓位的交易量和价位颜色)。

```
color  ColorVolumes() const
```

返回值

已分配到类实例中的对象"ColorVolumes"属性值。如果没有已分配图表，则返回 CLR_NONE。

## ColorVolumes (Set方法)

设置"ColorVolumes"属性的新值。

```
bool  ColorVolumes(
   color  new_color      // 交易量新颜色 (开仓价位)
   )
```

参数

*new_color*

[输入] 交易量新颜色 (开仓价位)。

返回值

true 如果成功，false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorLineBid (Get方法)

获取"ColorLineBid"属性的值(卖价线颜色)。

```
color  ColorLineBid() const
```

返回值

已分配到类实例中的对象"ColorLineBid"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorLineBid (Set方法)

设置"ColorLineBid"属性的新值。

```
bool  ColorLineBid(
   color  new_color      // 新卖价线颜色
   )
```

参数

*new_color*

[输入]新卖价线颜色。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorLineAsk (Get方法)

获取"ColorLineAsk"属性的值(买价线颜色)。

```
color  ColorLineAsk() const
```

返回值

已分配到类实例中的对象"ColorLineAsk"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorLineAsk (Set方法)

设置"ColorLineAsk"属性的新值。

```
bool  ColorLineAsk(
   color  new_color      // 新买价线颜色
   )
```

参数

*new_color*

[输入]新买价线颜色。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorLineLast (Get 方法)

获取 "ColorLineLast" 属性的值 (最后一笔成交价位颜色)。

```
color  ColorLineLast() const
```

返回值

已分配到类实例中的对象 "ColorLineLast" 属性值。如果没有已分配图表，则返回 CLR_NONE。

## ColorLineLast (Set 方法)

设置 "ColorLineLast" 属性的新值。

```
bool  ColorLineLast(
   color  new_color      // 最后一笔成交价位的新颜色
   )
```

参数

*new_color*

[输入] 最后一笔成交价位的新颜色。

返回值

true 如果成功，false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

## ColorStopLevels (Get方法)

获取"ColorStopLevels"属性的值(止损和止盈位的颜色)。

```
color  ColorStopLevels() const
```

返回值

已分配到类实例中的对象"ColorStopLevels"属性值。如果没有已分配图表,则返回 CLR_NONE。

## ColorStopLevels (Set方法)

设置"ColorStopLevels"属性的新值。

```
bool  ColorStopLevels(
   color  new_color      // 止损和止盈位的新颜色
   )
```

参数

*new_color*

[输入]止损和止盈位的新颜色。

返回值

true 如果成功,false 如果颜色未改变。

© 2000-2025, MetaQuotes Ltd.

# VisibleBars

获取图表可视柱线总数。

```
int  VisibleBars() const
```

返回值

获取已分配到类实例中的图表可视柱线总数。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# WindowsTotal

获取图表窗口总数，包括图表指标的子窗口。

```
int  WindowsTotal() const
```

返回值

已分配到类实例中的图表窗口总数，包括图表指标的子窗口。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# WindowIsVisible

获取指定图表窗口的可视性标志。

```
bool  WindowIsVisible(
   int  num      // 子窗口号码
   ) const
```

参数

*num*

[输入] 子窗口号码 (0 表示基准窗口)。

返回值

返回已分配到图表实例中的指定图表窗口的可视性标志。如果没有已分配图表，则返回 false。

© 2000-2025, MetaQuotes Ltd.

# WindowHandle

获取图表的窗口句柄 (HWND)。

```
int  WindowHandle() const
```

返回值

已分配到图表实例的窗口句柄。如果没有已分配图表，则返回 INVALID_HANDLE。

© 2000-2025, MetaQuotes Ltd.

# FirstVisibleBar

获取图表的首根可视柱线号码。

```
int  FirstVisibleBar() const
```

返回值

已分配到图表实例中的首根可视柱线号码。如果没有已分配图表，则返回 -1。

© 2000-2025, MetaQuotes Ltd.

# WidthInBars

获取窗口宽度，单位为柱线。

```
int  WidthInBars() const
```

返回值

已分配到图表实例中的窗口宽度，单位柱线。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# WidthInPixels

获取子窗口宽度，单位为像素。

```
int  WidthInPixels() const
```

返回值

已分配到图表实例中的子窗口宽度，单位像素。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# HeightInPixels

获取子窗口高度，单位为像素。

```
int  HeightInPixels(
   int  num      // 子窗口号码
   ) const
```

参数

*num*

[输入] 子窗口号码（0 表示基准窗口）。

返回值

已分配到图表实例中的子窗口高度，单位像素。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# PriceMin

获取指定子窗口的最小价格。

```
double  PriceMin(
   int  num      // 子窗口号码
   ) const
```

参数

*num*

[输入] 子窗口号码 (0 表示基准窗口)。

返回值

已分配到类实例中的图表最小价格属性值。如果无分配对象，则返回 EMPTY_VALUE。

© 2000-2025, MetaQuotes Ltd.

# PriceMax

获取指定子窗口的最大价格。

```
double  PriceMax(
   int  num      // 子窗口号码
   ) const
```

参数

*num*

[输入] 子窗口号码 (0 表示基准窗口)。

返回值

已分配到类实例中的图表最大价格属性值。如果无分配对象，则返回 EMPTY_VALUE。

© 2000-2025, MetaQuotes Ltd.

## Attach

将当前图表分配到类实例。

```
void  Attach()
```

## Attach

将指定图表分配到类实例。

```
void  Attach(
   long  chart       // 图表标识符
   )
```

参数

*chart*

[输入] 用于分配的图表标识符。

© 2000-2025, MetaQuotes Ltd.

# FirstChart

将客户端的首个图表分配到类实例。

```
void  FirstChart()
```

© 2000-2025, MetaQuotes Ltd.

# NextChart

将客户端的下一个图表分配到类实例。

```
void  NextChart()
```

© 2000-2025, MetaQuotes Ltd.

# Open

以指定参数打开图表并将其分配到类实例。

```
long  Open(
   const string      symbol_name,      // 品名
   ENUM_TIMEFRAMES   timeframe         // 周期
   )
```

参数

*symbol_name*

[输入] 品名。NULL 代表当前图表的品名 (已由程序加载)。

*timeframe*

[输入] 图表时间帧 (ENUM_TIMEFRAMES 枚举)。0 代表当前时间帧。

返回值

图表标识符。

© 2000-2025, MetaQuotes Ltd.

# Detach

从类实例中分离图表。

```
void  Detach()
```

© 2000-2025, MetaQuotes Ltd.

# Close

关闭已分配到类实例的图表。

```
void  Close()
```

© 2000-2025, MetaQuotes Ltd.

# BringToTop

在其它图表之上显示图表。

```
bool  BringToTop() const
```

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# EventObjectCreate

设置发送通知标志，在图表上创建新对象事件 触发时发送给所有 MQL5 程序。

```
bool  EventObjectCreate(
   bool  flag      // 标志
   )
```

参数

*flag*

[输入] 新标志值。

返回值

true -如果成功，false - 如果标志未改变。

© 2000-2025, MetaQuotes Ltd.

# EventObjectDelete

设置发送通知标志，在图表上删除对象 事件 触发时发送给所有 MQL5 程序。

```
bool  EventObjectDelete(
   bool  flag      // 标志
   )
```

参数

*flag*

[输入] 新标志值。

返回值

true -如果成功，false - 如果标志未改变。

© 2000-2025, MetaQuotes Ltd.

# IndicatorAdd

添加指定句柄的指标到指定的图表窗口。

```
bool  IndicatorAdd(
   int   sub_win          // 子窗口号码
   int   handle           // 指标句柄
   );
```

参数

*sub_win*

[输入] 图表子窗口的号码。0 代表图表的主窗口。如果指定了一个不存在的窗口号码，则将创建一个新窗口。

*handle*

[输入] 指标句柄。

返回值

函数在成功时返回 true，否则返回 false。为获得有关 出错信息，请调用 GetLastError() 函数。

参阅

IndicatorDelete(), IndicatorsTotal(), IndicatorName()。

© 2000-2025, MetaQuotes Ltd.

# IndicatorDelete

从指定的图表窗口删除指定名称的指标。

```
bool  IndicatorDelete(
   int            sub_win        // 子窗口号码
   const string   name           // 指标简称
   );
```

参数

*sub_win*

[输入] 图表子窗口的号码。0 代表图表的主窗口。

*const name*

[输入] 指标的简称，即在 INDICATOR_SHORTNAME 属性里，使用 IndicatorSetString() 函数设置的值。为获得指标的简称请使用 IndicatorName() 函数。

返回值

指标成功删除后返回 true。否则返回 false。为获得 出错信息 详情请使用 GetLastError() 函数。

注释

如果在同一图表子窗口里存在两个相同简称标识符的指标，则第一个被删除。

如果图表中有其它指标基于此被删除的指标值，则这些指标也一并删除。

请不要将指标简称与使用函数 iCustom() 和 IndicatorCreate() 创建指标时指定的文件名混淆。如果指标的简称没有明确设置，则在编译期间用包含指标源代码的文件名来指定。

从图表中删除一个指标并不代表其计算部分也从临时内存里删除。释放指标句柄请使用 IndicatorRelease() 函数。

指标的简称应规范正确。它应该在 INDICATOR_SHORTNAME 属性里使用 IndicatorSetString() 函数来写入。建议指标简称应包含所有输入参数值，因为从图表里用 IndicatorDelete() 函数删除指标时将以简称作为标识。

参阅

IndicatorAdd(), IndicatorsTotal(), IndicatorName(), iCustom(), IndicatorCreate(), IndicatorSetString()。

© 2000-2025, MetaQuotes Ltd.

# IndicatorsTotal

返回应用于指定图表子窗口中的所有指标数量。

```
int  IndicatorsTotal(
   int   sub_win     // 子窗口号码
   );
```

参数

*sub_win*

[输入] 图表子窗口的号码。0 代表图表的主窗口。

返回值

指定图表子窗口中的指标数量。为获得 出错信息 详情请使用 GetLastError() 函数。

注释

函数允许通过所有挂载到图表上的指标进行搜索。图表的所有窗口数量可从 CHART_WINDOWS_TOTAL 属性里使用 GetInteger() 函数获得。

参阅

IndicatorAdd(), IndicatorDelete(), IndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()。

© 2000-2025, MetaQuotes Ltd.

# IndicatorName

返回在指定图表窗口上的指标列表里由索引值定位的指标简称。

```
string  IndicatorName(
   int   sub_win     // 子窗口号码
   int   index       // 添加在图表子窗口上的指标列表中的索引值
   );
```

参数

*sub_win*

[输入] 图表子窗口的号码。0 代表图表的主窗口。

*index*

[输入] 指标列表中的指标索引值。指标号码自零开始计数，即列表中的第一个指标索引为 0。为得到列表中的指标数量请使用 IndicatorsTotal() 函数。

返回值

指标的简称，即在 INDICATOR_SHORTNAME 属性里，使用 IndicatorSetString() 函数设置的值。为获得 出错信息 详情请使用 GetLastError() 函数。

注释

请不要将指标简称与使用函数 iCustom() 和 IndicatorCreate() 创建指标时指定的文件名混淆。如果指标的简称没有明确设置，则在编译期间用包含指标源代码的文件名来指定。

指标的简称应规范正确。它应该在 INDICATOR_SHORTNAME 属性里使用 IndicatorSetString() 函数来写入。建议指标简称应包含所有输入参数值，因为从图表里用 IndicatorDelete() 函数删除指标时将以简称作为标识。

参阅

IndicatorAdd(), IndicatorDelete, IndicatorsTotal, iCustom(), IndicatorCreate(), IndicatorSetString()。

© 2000-2025, MetaQuotes Ltd.

# Navigate

遍历图表。

```
bool  Navigate(
   ENUM_CHART_POSITION  position,     // 位置
   int                  shift=0       // 偏移
   )
```

参数

*position*

[输入] 值为 ENUM_CHART_POSITION 枚举。

*shift=0*

[输入] 偏移柱线数。

返回值

true 如果成功，false 如果图表不可遍历。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获得图表品名。

```
string  Symbol() const
```

返回值

已分配到类实例的图表品名。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# Period

获得图表周期。

```
ENUM_TIMEFRAMES  Period() const
```

返回值

已分配到类实例的图表周期。如果没有任何已分配图表，则返回 0。

© 2000-2025, MetaQuotes Ltd.

# 重绘

重绘已分配到类实例的图表。

```
void  Redraw()
```

© 2000-2025, MetaQuotes Ltd.

# GetInteger

此函数返回相应的对象属性值。对象属性必须是整数类型。函数有两个变种。

1. 立即返回属性值。

```
long  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,          // 属性标识符
   int                          sub_window=0      // 子窗口号码
   ) const
```

2. 如果成功，属性值置于指定的整数类型变量，传递引用作为最后的参数。

```
bool  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,          // 属性标识符
   int                          sub_window,       // 子窗口号码
   long&                        value             // 此处我们得到属性值
   ) const
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_INTEGER 枚举)。

*sub_window*

[输入] 图表子窗口号码。

*value*

[输入] 整数类型变量用于接收请求属性的值。

返回值

已分配到类实例中的图表属性值。如果无分配图表，则返回 -1。

对于第二个函数变种，如果该属性已赋值到变量，则返回 true，否则返回 false。为读取更多有关 出错信息 请调用 GetLastError()。

© 2000-2025, MetaQuotes Ltd.

# SetInteger

设置整数类型属性新数值。

```
bool  SetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,      // 属性标识符
   long                         value         // 新值
   )
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_INTEGER 枚举)。

*value*

[输入] 属性的新值。

返回值

true 如果成功, false 如果整数类型的属性未改变。

© 2000-2025, MetaQuotes Ltd.

# GetDouble

此函数返回相应的对象属性值。对象属性必须是双精度类型。函数有两个变种。

1. 立即返回属性值。

```
double  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,          // 属性标识符
   int                         sub_window=0    // 子窗口号码
   ) const
```

2. 如果成功，属性值置于指定的双精度类型变量，传递引用作为最后的参数。

```
bool  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,        // 属性标识符
   int                         sub_window,     // 子窗口号码
   double&                     value           // 此处我们得到属性值
   ) const
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_DOUBLE 枚举)。

*sub_window*

[输入] 图表子窗口号码。

*value*

[输入] 双精度类型变量用于接收请求属性的值。

返回值

已分配到类实例中的图表属性值。如果无分配图表，则返回 EMPTY_VALUE。

对于第二个函数变种，如果该属性已赋值到变量，则返回 true，否则返回 false。为读取更多有关 出错信息 请调用 GetLastError()。

© 2000-2025, MetaQuotes Ltd.

# SetDouble

设置双精度类型属性新数值。

```
bool  SetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,      // 属性标识符
   double                      value         // 新值
   )
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_DOUBLE 枚举)。

*value*

[输入] 属性的新值。

返回值

true 如果成功, false 如果双精度类型的属性未改变。

© 2000-2025, MetaQuotes Ltd.

# GetString

此函数返回相应的对象属性值。对象属性必须是字符串类型。函数有两个变种。

1. 立即返回属性值。

```
string  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id      // 属性标识符
   ) const
```

2. 如果成功，属性值置于指定的字符串类型变量，传递引用作为最后的参数。

```
bool  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,     // 属性标识符
   string&                     value        // 此处我们得到属性值
   ) const
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_STRING 枚举)。

*sub_window*

[输入] 图表子窗口号码。

*value*

[输入] 字符串类型变量用于接收请求属性的值。

返回值

已分配到类实例中的图表属性值。如果无分配图表，则返回 ""。

对于第二个函数变种，如果该属性已赋值到变量，则返回 true，否则返回 false。为读取更多有关 出错信息 请调用 GetLastError()。

© 2000-2025, MetaQuotes Ltd.

# SetString

设置字符类型属性新值。

```
bool  SetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,      // 属性标识符
   string                      value         // 新属性值
   )
```

参数

*prop_id*

[输入] 属性标识符 (ENUM_CHART_PROPERTY_STRING 枚举)。

*value*

[输入] 属性的新值。

返回值

true 如果成功，false 如果字符串类型的属性未改变。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

改变已分配到类实例的图表的品名和周期。

```
bool  SetSymbolPeriod(
   const string      symbol_name,      // 品名
   ENUM_TIMEFRAMES   timeframe         // 周期
   )
```

参数

*symbol_name*

[输入] 新品名。NULL 代表当前图表的品名 (已由程序加载)。

*timeframe*

[输入] 新图表时间帧 (ENUM_TIMEFRAMES 枚举)。0 代表当前时间帧。

返回值

true 如果成功, false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# ApplyTemplate

应用指定模板至图表。

```
bool  ApplyTemplate(
   const string  filename      // 模板文件名称
   )
```

参数

*filename*

[输入] 模板文件的名称。

返回值

true 如果成功，false 如果模板未能应用。

© 2000-2025, MetaQuotes Ltd.

# ScreenShot

创建指定图表的截屏并将之保存为 .gif 文件。

```
bool  ScreenShot(
   string           filename,                  // 文件名
   int              width,                     // 宽度
   int              height,                    // 高度
   ENUM_ALIGN_MODE  align_mode=ALIGN_RIGHT     // 对齐类型
   ) const
```

参数

*filename*

[输入] 截屏文件名。

*width*

[输入] 截屏宽度，单位像素。

*height*

[输入] 截屏高度，单位像素。

*align_mode=ALIGN_RIGHT*

[输入] 若截屏太窄，对齐模式。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# WindowOnDropped

获取与对象（智能程序或脚本）落点相应的图表子窗口号码

```
int  WindowOnDropped() const
```

返回值

对象落点的图表子窗口号码。0 代表图表主窗口。

© 2000-2025, MetaQuotes Ltd.

# PriceOnDropped

获取与对象（智能程序或脚本）落点相应的价格坐标。

```
double  PriceOnDropped() const
```

返回值

对象落点的价格坐标。

© 2000-2025, MetaQuotes Ltd.

# TimeOnDropped

获取与对象（智能程序或脚本）落点相应的时间坐标。

```
datetime  TimeOnDropped() const
```

返回值

对象落点的时间坐标。

© 2000-2025, MetaQuotes Ltd.

# XOnDropped

获取与对象（智能程序或脚本）落点相应的 X 坐标。

```
int  XOnDropped() const
```

返回值

对象落点的 X 坐标。

© 2000-2025, MetaQuotes Ltd.

# YOnDropped

获取与对象（智能程序或脚本）落点相应的 Y 坐标。

```
int  YOnDropped() const
```

返回值

对象落点的 Y 坐标。

© 2000-2025, MetaQuotes Ltd.

# Save

保存对象参数至文件。

```
virtual bool  Save(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen(...) 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件里加载对象参数。

```
virtual bool  Load(
   int  file_handle      // 文件句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄，文件已由 FileOpen(...) 函数打开。

返回值

true 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# Type

返回图形对象类型标识符。

```
virtual int  Type() const
```

返回值

对象类型标识符 (对于 CChart 为 0x1111)。

© 2000-2025, MetaQuotes Ltd.

# 图形

图形程序库包含快速绘制自定义图表的类和全局函数。程序库为建立坐标轴，曲线以及函数类提供了便捷的现有解决方案以快速访问改变自定义图表的常规属性。

若要开始使用该函数库，只需阅读文章想象一下！MQL5图形库类似于R语言的'plot'。

图形程序库位于程序端工作目录的Include\Graphics文件夹。

| 类 | 描述 |
| --- | --- |
| CAxis | 处理坐标轴的类 |
| ColorGenerator | 指定默认配色方案的类 |
| CCurve | 处理曲线的类 |
| CGraphic | 创建自定义图表的基本类 |

© 2000-2025, MetaQuotes Ltd.

# GraphPlot

快速绘制曲线的函数。

使用Y坐标绘制一条曲线的版本。

```
string  GraphPlot(
   const double      &y[],                  // Y坐标
   ENUM_CURVE_TYPE   type=CURVE_POINTS      // 曲线类型
   )
```

注意

Y数组索引用作曲线的X坐标。

使用X和Y坐标绘制一个曲线的版本

```
string  GraphPlot(
   const double      &x[],                  // X 坐标
   const double      &y[],                  // Y坐标
   ENUM_CURVE_TYPE   type=CURVE_POINTS      // 曲线类型
   )
```

使用X和Y坐标绘制两条曲线的版本

```
string  GraphPlot(
   const double      &x1[],                 // X 坐标
   const double      &y1[],                 // Y 坐标
   const double      &x2[],                 // X 坐标
   const double      &y2[],                 // Y 坐标
   ENUM_CURVE_TYPE   type=CURVE_POINTS      // 曲线类型
   )
```

使用X和Y坐标绘制三条曲线的版本

```
string  GraphPlot(
   const double      &x1[],                 // X 坐标
   const double      &y1[],                 // Y 坐标
   const double      &x2[],                 // X 坐标
   const double      &y2[],                 // Y 坐标
   const double      &x3[],                 // X 坐标
   const double      &y3[],                 // Y 坐标
   ENUM_CURVE_TYPE   type=CURVE_POINTS      // 曲线类型
   )
```

© 2000-2025, MetaQuotes Ltd.

使用CPoint2D点坐标绘制一条曲线的版本

```
string  GraphPlot(
   const CPoint2D   &points[],            // 曲线坐标
   ENUM_CURVE_TYPE  type=CURVE_POINTS     // 曲线类型
   )
```

使用CPoint2D点坐标绘制两条曲线的版本

```
string  GraphPlot(
   const CPoint2D   &points1[],           // 曲线坐标
   const CPoint2D   &points2[],           // 曲线坐标
   ENUM_CURVE_TYPE  type=CURVE_POINTS     // 曲线类型
   )
```

使用CPoint2D点坐标绘制三条曲线的版本

```
string  GraphPlot(
   const CPoint2D   &points1[],           // 曲线坐标
   const CPoint2D   &points2[],           // 曲线坐标
   const CPoint2D   &points3[],           // 曲线坐标
   ENUM_CURVE_TYPE  type=CURVE_POINTS     // 曲线类型
   )
```

使用CurveFunction函数指针绘制一条曲线的版本

```
string  GraphPlot(
   CurveFunction    function,             // 函数指针
   const double     from,                 // 参数初始值
   const double     to,                   //参数最终值
   const double     step,                 // 参数盈余
   ENUM_CURVE_TYPE  type=CURVE_POINTS     // 曲线类型
   )
```

使用CurveFunction函数指针绘制两条曲线的版本

```
string  GraphPlot(
   CurveFunction    function1,            // 函数指针
   CurveFunction    function2,            // 函数指针
   const double     from,                 // 参数初始值
   const double     to,                   //参数最终值
   const double     step,                 // 参数盈余
   ENUM_CURVE_TYPE  type=CURVE_POINTS     // 曲线类型
   )
```

© 2000-2025, MetaQuotes Ltd.

使用CurveFunction函数指针绘制三条曲线的版本

```
string  GraphPlot(
   CurveFunction    function1,              // 函数指针
   CurveFunction    function2,              // 函数指针
   CurveFunction    function3,              // 函数指针
   const double     from,                   // 参数初始值
   const double     to,                     //参数最终值
   const double     step,                   // 参数盈余
   ENUM_CURVE_TYPE  type=CURVE_POINTS       // 曲线类型
   )
```

参数

*&x[]*

[in] X坐标。

*&y[]*

[in] Y坐标。

*&x1[]*

[in] 第一条曲线的X坐标。

*&y1[]*

[in] 第一条曲线的Y坐标。

*&x2[]*

[in] 第二条曲线的X坐标。

*&y2[]*

[in] 第二条曲线的Y坐标。

*&x3[]*

[in] 第三条曲线的X坐标。

*&y3[]*

[in] 第三条曲线的Y坐标。

*&points[]*

[in] 曲线点坐标。

*&points1[]*

[in] 第一条曲线点的坐标。

*&points2[]*

[in] 第二条曲线点的坐标。

*&points3[]*

[in] 第三条曲线点的坐标。

© 2000-2025, MetaQuotes Ltd.

*function*
[in] CurveFunction函数的指针。

*function1*
[in] 第一个函数的指针。

*function2*
[in] 第二个函数的指针。

*function3*
[in] 第三个函数的指针。

*from*
[in] 对应第一个 X 坐标。

*to*
[in] 对应最后一个X坐标。

*step*
[in] 计算X坐标的参数。

*type=CURVE_POINTS*
[in] 曲线类型。

返回值

图形资源名称。

© 2000-2025, MetaQuotes Ltd.

# CAxis

CAxis是用于处理坐标轴的辅助图形库类。

## 描述

CAxis类可以接收和存储坐标轴的不同参数。该类实施自动动态缩放坐标轴的能力。

## 声明

```
class CAxis
```

## 主题

```
#include <Graphics\Axis.mqh>
```

## 类函数

| Method | 描述 |
| --- | --- |
| AutoScale | 获得/设置自动缩放标识 |
| Min | 获得/设置坐标轴的最小值 |
| Max | 获得/设置坐标轴的最大值 |
| Step | 获得坐标轴的步骤值 |
| Name | 获得/设置坐标轴名称 |
| Color | 获得/设置坐标轴颜色 |
| ValuesSize | 获得/设置坐标轴数的大小 |
| ValuesWidth | 获得/设置坐标轴显示的最长数字 |
| ValuesFormat | 获得/设置坐标轴数字的格式 |
| ValuesDateTimeMode | 获得将日期转为字符串的格式。 |
| ValuesFunctionFormat | 获得定义坐标轴显示值格式的函数指针 |
| ValuesFunctionFormatCBData | 获得可能包含有关转换坐标轴值额外数据的对象指针。 |
| NameSize | 获得/设置坐标轴名称的字体大小 |
| ZeroLever | 获得/设置 “零”值 |
| DefaultStep | 获得/设置坐标轴的初始步骤值 |
| MaxLabels | 获得/设置坐标轴上的最大数字量 |
| MinGrace | 获得/设置坐标轴的最小 “公差”值 |
| MaxGrace | 获得/设置坐标轴的最大 “公差”值 |
| SelectAxisScale | 自动缩放坐标轴。 |

© 2000-2025, MetaQuotes Ltd.

## AutoScale（获得方法）

返回定义自动比例需求的标识

```
bool AutoScale()
```

返回值

标识值。

注意

true —执行自动比例。

false —不执行自动比例。

## AutoScale（设置方法）

设置定义自动比例需求的标识。

```
void  AutoScale(
   const bool  auto      // 标识值
   )
```

参数

*auto*

[in]

注意

true —执行自动比例。

false —不执行自动比例。

© 2000-2025, MetaQuotes Ltd.

## Min (获得方法)

返回最低轴值。

```
double  Min()
```

返回值

最大轴值。

## Min (设置方法)

设置最低轴值。

```
void  Min(
   const double  min       // 最底轴值
   )
```

参数

*min*

[in] 最底轴值。

© 2000-2025, MetaQuotes Ltd.

## Max（获得方法）

返回最大轴值。

```
double  Max()
```

返回值

最大轴值。

## Max（设置方法）

设置最大轴值。

```
void  Max(
   const double  max       // 最大轴值
   )
```

参数

*max*

[in] 最大轴值。

© 2000-2025, MetaQuotes Ltd.

# Step（获得方法）

返回坐标轴的步骤值。

```
double   Step()
```

返回值

步骤值。

© 2000-2025, MetaQuotes Ltd.

## Name（获得方法）

返回坐标轴名称。

```
string  Name()
```

返回值

坐标轴名称。

## Name（设置方法）

设置坐标轴名称。

```
void  Name(
   const string  name      //坐标轴名称
   )
```

参数

*name*

[in] 坐标轴名称。

© 2000-2025, MetaQuotes Ltd.

## Color (获取方法)

返回坐标轴颜色。

```
color  Color()
```

返回值

坐标轴颜色。

## Color (设置方法)

设置坐标轴颜色

```
void  Color(
   const color  clr      //坐标轴颜色
   )
```

参数

*clr*

[in] 坐标轴颜色。

© 2000-2025, MetaQuotes Ltd.

## ValuesSize (获得方法)

返回坐标轴数字的大小。

```
int  ValuesSize()
```

返回值

坐标轴数字的大小。

## ValuesSize (设置方法)

设置坐标轴数字的大小。

```
void  ValuesSize(
   const int  size      //坐标轴数字的大小
   )
```

参数

*size*

[in] 坐标轴数字的大小

© 2000-2025, MetaQuotes Ltd.

# ValuesWidth（获得方法）

返回以像素为单位允许显示的最长坐标轴数字。

```
int  ValuesWidth()
```

返回值

以像素为单位的坐标轴数字的长度。

注意

如果以像素为单位指定数的长度超出了允许显示的最大长度，则会将其缩短并以缩写点结束。

# ValuesWidth（设置方法）

设置以像素为单位允许显示的最长坐标轴数字。

```
void  ValuesWidth(
   const int  width      // 以像素为单位允许显示的最大长度
   )
```

参数

*宽度*

[in] 坐标轴数字允许的最大长度。

注意

如果以像素为单位指定数的长度超出了允许显示的最大长度，则会将其缩短并以缩写点结束。

© 2000-2025, MetaQuotes Ltd.

## ValuesFormat（获得方法）

返回坐标轴数字的格式。

```
string  ValuesFormat()
```

返回值

数字格式。

## ValuesFormat（设置方法）

设置坐标轴数字的格式。

```
void  ValuesFormat(
   const string  format      //坐标轴数字的格式
   )
```

参数

*格式*

[in] 坐标轴数字的格式。

© 2000-2025, MetaQuotes Ltd.

## NameSize (获得方法)

返回坐标轴名称的字体大小。

```
int  NameSize()
```

返回值

坐标轴名称的字体大小。

## NameSize (设置方法)

设置坐标轴名称的字体大小。

```
void  NameSize(
   const int  size      // 坐标轴名称的字体大小
   )
```

参数

*size*

[in] 坐标轴名称的字体大小。

© 2000-2025, MetaQuotes Ltd.

## ZeroLever (获得方法)

返回 "零基准"值。

```
double  ZeroLever()
```

返回值

"零基准"。

注意

当坐标轴比例范围应该扩展到包括零值时使用该值来定义。

## ZeroLever (设置方法)

设置 "零基准"值。

```
void  ZeroLever(
   const double  value      //  "零基准"值
   )
```

参数

*值*

[in] "零基准"值。

注意

当坐标轴比例范围应该扩展到包括零值时使用该值来定义。

© 2000-2025, MetaQuotes Ltd.

## DefaultStep（获取方法）

返回坐标轴的初始步骤值

```
double  DefaultStep()
```

返回值

坐标轴的步骤。

## DefaultStep（设置方法）。

设置坐标轴的初始步骤值

```
void  DefaultStep(
   const double  value      //坐标轴的步骤
   )
```

参数

*值*

[in] 坐标轴的初始步骤值。

© 2000-2025, MetaQuotes Ltd.

## MaxLabels (获得方法)

返回坐标轴上可以显示的数字最大值

```
double  MaxLabels()
```

返回值

坐标轴上的数字最大值

## MaxLabels (设置方法)

设置坐标轴上可以显示的数字最大值。

```
void  MaxLabels(
   const double  value      // 最大数
   )
```

参数

*值*

[in] 坐标轴上可以显示的数字最大值

© 2000-2025, MetaQuotes Ltd.

# MinGrace (获得方法)

返回应用到最低轴值的 "公差"。

```
double  MinGrace()
```

返回值

最低轴值的 "公差"值。

注意

该值表示为坐标轴整体长度的一部分。例如，假设轴值处于4.0-16.0之间，那么其长度即是12.0。如果MinGrace等于0.1，那么会从最低轴值减去坐标轴长度的10% (或1.2)。因此，坐标轴涵盖2.8 -16.0的区间。

# MinGrace (设置方法)

设置应用到最低轴值的 "公差"。

```
void  MinGrace(
   const double  value      //  "公差"值
   )
```

参数

*值*

[in] 应用到最低轴值的 "公差"值。

注意

该值表示为坐标轴整体长度的一部分。例如，假设轴值处于4.0-16.0之间，那么其长度即是12.0。如果MinGrace等于0.1，那么会从最低轴值减去坐标轴长度的10% (或1.2)。因此，坐标轴涵盖2.8 -16.0的区间。

© 2000-2025, MetaQuotes Ltd.

# MaxGrace (获得方法)

返回应用到最大轴值的 "公差"。

```
double  MaxGrace()
```

返回值

最大轴值的 "公差"值。

注意

该值表示为坐标轴整体长度的一部分。例如，假设轴值处于4.0-16.0之间，那么其长度即是12.0。如果MaxGrace等于0.1，那么坐标轴长度的10% (或1.2)会添加到最大轴值。 因此，坐标轴涵盖4.0-17.2的区间。

# MaxGrace (设置方法)

设置应用到最大轴值的 "公差"。

```
void  MaxGrace(
   const double  value      //  "公差"值
   )
```

参数

*值*

[in] 应用到最大轴值的 "公差"值。

注意

该值表示为坐标轴整体长度的一部分。例如，假设轴值处于4.0-16.0之间，那么其长度即是12.0。如果MinGrace等于0.1，那么会从最低轴值减去坐标轴长度的10% (或1.2)。 因此，坐标轴涵盖2.8 -16.0的区间。

© 2000-2025, MetaQuotes Ltd.

# ValuesDateTimeMode (Метод Get)

获得将日期转为字符串的格式。

```
int  ValuesDateTimeMode()
```

返回值

将日期转为字符串的格式。

# ValuesDateTimeMode (Метод Set)

设置将日期转为字符串的格式。

```
void  ValuesDateTimeMode(
   const int  mode      // 将日期转为字符串的格式
   )
```

参数

*mode*

[in] 转换格式。

注意

在 TimeToString()函数描述中了解更多有关将日期转换为字符串的格式。

© 2000-2025, MetaQuotes Ltd.

## ValuesFunctionFormat（获得方法）

获得定义坐标轴显示值格式的函数指针

```
DoubleToStringFunction  ValuesFunctionFormat()
```

返回值

定义坐标轴显示值格式的函数指针。

## ValuesFunctionFormat（设置方法）

设置定义坐标轴显示值格式的函数指针。

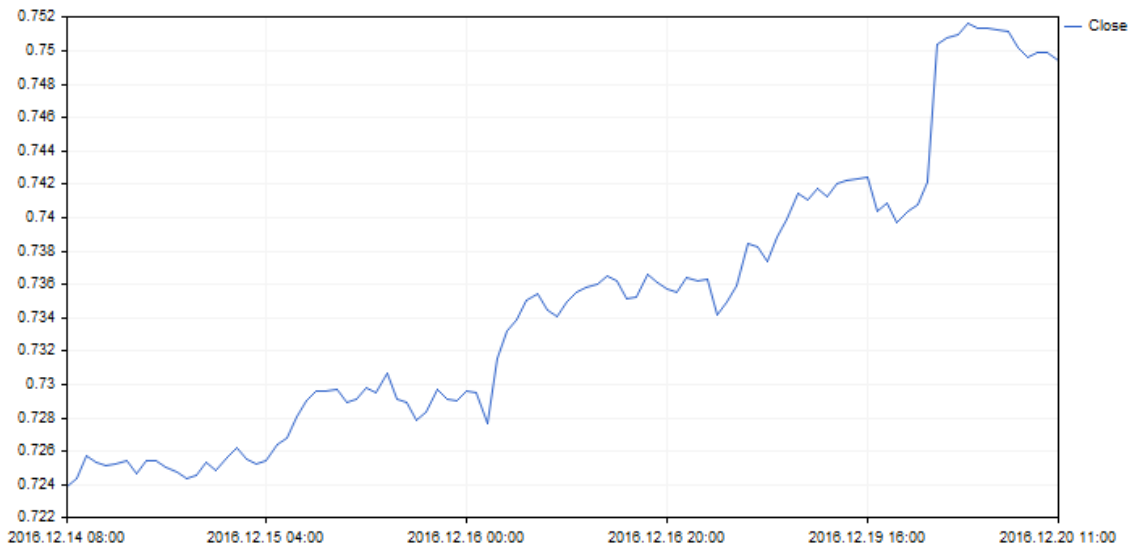
```
void  ValuesFunctionFormat(
   DoubleToStringFunction  func      // 将数值转为字符串的函数
   )
```

参数

*func*

[in] 将数值转为字符串的自定义函数。

示例：




通过以下代码改变显示X坐标轴值的格式：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                DateAxisGraphic.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//--- array for store values
double arrX[];
double arrY[];
//+------------------------------------------------------------------+
//| 创建X轴值的自定义函数                                              |
//+------------------------------------------------------------------+
string TimeFormat(double x,void *cbdata)
  {
   return(TimeToString((datetime)arrX[ArraySize(arrX)-(int)x-1]));
  }
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlRates rates[];
   CopyRates(Symbol(),Period(),0,100,rates);
   ArraySetAsSeries(rates,true);
   int size=ArraySize(rates);
   ArrayResize(arrX,size);
   ArrayResize(arrY,size);
   for(int i=0; i<size;++i)
     {
      arrX[i]=(double)rates[i].time;
      arrY[i]=rates[i].close;
     }
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"DateAxisGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"DateAxisGraphic");
     }
//--- create curve
   CCurve *curve=graphic.CurveAdd(arrY,CURVE_LINES);
//--- 获得X轴
   CAxis *xAxis=graphic.XAxis();
//--- 设置 X 轴属性
   xAxis.AutoScale(false);
   xAxis.Type(AXIS_TYPE_CUSTOM);
   xAxis.ValuesFunctionFormat(TimeFormat);
   xAxis.DefaultStep(20.0);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

## ValuesFunctionFormatCBData（获得方法）

获得可能包含有关转换坐标轴值额外数据的对象指针。

```
void*  ValuesFunctionFormatCBData()
```

返回值

可能包含有关转换坐标轴值额外数据的对象指针。

## ValuesFunctionFormatCBData（设置方法）

设置可能包含有关转换坐标轴值额外数据的类对象指针。

```
void  ValuesFunctionFormatCBData(
   void*  cbdata      // 类对象指针
   )
```

参数

*cbdata*

[in] 包含有关转换坐标轴值额外数据的类对象指针

© 2000-2025, MetaQuotes Ltd.

# SelectAxisScale

自动缩放坐标轴。

```
void  SelectAxisScale()
```

© 2000-2025, MetaQuotes Ltd.

# CColorGenerator

CColorGenerator类是一种用来处理调色板的辅助的图形程序类。

## 描述

CColorGenerator类包含用于曲线的默认初始调色板（如果用户未指定颜色）。

如果已使用初始调色板中的全部颜色，那么将会自动生成新颜色并且调色板会被重新充满。

## 声明

```
class CColorGenerator
```

## 主题

```
#include <Graphics\ColorGenerator.mqh>
```

## 类方法

| 方法 | 描述 |
| --- | --- |
| Next | 返回调色板的下一个颜色 |
| Reset | 重置生成器 |

© 2000-2025, MetaQuotes Ltd.

# Next

返回调色板的下一个颜色。

```
uint  Next()
```

返回值

颜色。

注意

如果已使用初始调色板中的全部颜色，那么将会自动生成新颜色并且调色板会被重新充满。

© 2000-2025, MetaQuotes Ltd.

# Reset

重置生成器。

```
void  Reset()
```

© 2000-2025, MetaQuotes Ltd.

# CCurve

CCurve类可以处理图表上生成的曲线属性。

## 描述

当处理CGraphic类时，CCurve类可以设置，安装和接收坐标和曲线的不同属性。

共有三种绘制曲线模式：点式，线型和直方图。在类中每个绘制模式可以实行独立的参数。

## 声明

```
class CCurve : public CObject
```

## 主题

```
#include <Graphics\Curve.mqh>
```

## 继承体系

CObject
  CCurve

## 类函数

| Method | 描述 |
|---|---|
| Type | 获得曲线类型 |
| Name | 获得曲线名称 |
| Color | 获得曲线颜色 |
| XMax | 获得X函数的最大值 |
| XMin | 获得X函数的最小值 |
| YMax | 获得Y函数的最大值 |
| YMin | 获得Y函数的最小值 |
| Size | 获得定义曲线的点数 |
| PointsSize | 获得/设置定义曲线的点的线性尺寸 |
| PointsFill | 获得/设置定义曲线的填充点的标识 |
| PointsColor | 获得/设置点填充颜色 |
| GetX | 获得数组的所有曲线点的值 |
| GetY | 获得数组的所有曲线点的值 |
| LinesStyle | 获得/设置绘制线型曲线的线型样式 |
| LinesIsSmooth | 获得/设置使用线型绘制时的平滑标识 |

© 2000-2025, MetaQuotes Ltd.

| Method | 描述 |
| --- | --- |
| LinesSmoothTension | 获得/设置使用线型绘制时的曲线平滑参数 |
| LinesSmoothStep | 获得/设置使用线型绘制时的平滑近似线的长度 |
| LinesWidth | 获得/设置使用线型绘制曲线时的线型宽度 |
| HistogramWidth | 获得/设置使用直方图绘制时的列宽 |
| CustomPlotCBData | 获得/设置自定义曲线绘制模式中使用的对象指针。 |
| CustomPlotFunction | 获得/设置实施自定义曲线绘制模式的函数指针。 |
| PointsType | 获得/设置绘制点式曲线时指明所用点式类型的标识。 |
| StepsDimension | 获得/设置指明步进式曲线渲染所使用的维度的值。 |
| TrendLineCoefficients | 获得/设置写入数组的趋势线比率。 |
| TrendLineColor | 获得/设置曲线趋势线的颜色。 |
| TrendLineVisible | 获得/设置趋势线可视标识。 |
| Update | 更新曲线坐标。 |
| Visible | 获得/设置定义函数在图表是否可见的标识。 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Compare

© 2000-2025, MetaQuotes Ltd.

# Type

返回曲线类型。

```
ENUM_CURVE_TYPE   Type()
```

返回值

曲线类型。

© 2000-2025, MetaQuotes Ltd.

# Name

返回曲线名称。

```
string  Name()
```

返回值

曲线名称。

© 2000-2025, MetaQuotes Ltd.

# Color

返回曲线颜色。

```
uint  Color()
```

返回值

曲线颜色。

© 2000-2025, MetaQuotes Ltd.

# XMax

返回X函数的最大值（仅实数）。

```
double   XMax()
```

返回值

全部函数参数中的最大实数。

© 2000-2025, MetaQuotes Ltd.

# XMin

返回X函数的最小值（仅实数）。

```
double   XMin()
```

返回值

全部函数参数中的最小实数。

© 2000-2025, MetaQuotes Ltd.

# YMax

返回Y函数的最大值（仅实数）。

```
double   YMax()
```

返回值

Y函数的最大值（仅实数）。

© 2000-2025, MetaQuotes Ltd.

# YMin

返回Y函数的最小值（仅实数）。

```
double   YMin()
```

返回值

Y函数的最小值（仅实数）。

© 2000-2025, MetaQuotes Ltd.

# Size

返回定义曲线的点数。

```
int  Size()
```

返回值

定义曲线的点数。

© 2000-2025, MetaQuotes Ltd.

## PointsSize (获得方法)

返回绘制曲线点使用的线性尺寸 (以像素为单位)。

```
int  PointsSize()
```

返回值

以像素为单位定义曲线点的尺寸。

## PointsSize (设置方法)

设置绘制曲线点使用的线性尺寸 (以像素为单位)。

```
void  PointsSize(
   const int  size      // 以像素为单位的点的尺寸
   )
```

参数

*size*

[in] 绘制曲线点使用的线性尺寸 (以像素为单位)。

© 2000-2025, MetaQuotes Ltd.

## PointsFill (获得方法)

如果应该执行填充定义曲线点，返回决定标识。

```
bool  PointsFill ()
```

返回值

标识值。

注意

true —执行填充

false —不执行填充

## PointsFill (设置方法)

如果应该执行填充定义曲线点，设置决定标识。

```
void  PointsFill(
   const bool  fill      // 标识值
   )
```

参数

*fill*

[in]标识值。

注意

true —执行填充

false —不执行填充

© 2000-2025, MetaQuotes Ltd.

## PointsColor (获得方法)

返回填充颜色点。

```
uint  PointsColor ()
```

返回值

填充定义曲线点的颜色。

## PointsColor (设置方法)

设置填充颜色点

```
void  PointsColor(
   const uint  clr      //填充颜色点
   )
```

参数

*clr*

[in]填充定义曲线点的颜色。

© 2000-2025, MetaQuotes Ltd.

# GetX

编写全部数组曲线点的X坐标。

```
void  GetX(
   double  &x[]      // X坐标
   )
```

参数

*&x[]*

[out] 编写X坐标的数组。

© 2000-2025, MetaQuotes Ltd.

# GetY

编写全部数组曲线点的Y坐标。

```
void  GetY(
   double  &y[]      // Y坐标
   )
```

参数

*&y[]*

[out] 编写Y坐标的数组。

© 2000-2025, MetaQuotes Ltd.

## LinesStyle (获得方法)

返回使用线性模式绘制曲线时的线性风格。

```
ENUM_LINE_STYLE  LinesStyle()
```

返回值

线性风格。

## LinesStyle (设置方法)

设置使用线性模式绘制曲线时的线性风格。

```
void  LinesStyle (
   ENUM_LINE_STYLE  style      // 线性风格
   )
```

参数

*style*

[in] 线性风格。

© 2000-2025, MetaQuotes Ltd.

## LinesIsSmooth（获得方法）

返回通过线性模式绘制曲线时如果定义平滑应该完成的标识。

```
bool  LinesIsSmooth()
```

返回值

标识值

注意

true —执行平滑

false —不执行平滑

## LinesIsSmooth（设置方法）

设置通过线性模式绘制曲线时如果定义平滑应该完成的标识。

```
void  LinesIsSmooth(
   const bool  smooth      //标识值
   )
```

参数

*smooth*

[in]标识值

注意

true —执行平滑

false —不执行平滑

© 2000-2025, MetaQuotes Ltd.

## LinesSmoothTension（获得方法）

返回使用线性绘制时的曲线平滑参数。

```
double  LinesSmoothTension()
```

返回值

平滑参数值

注意

'张力'值的范围是(0.0; 1.0]。

## LinesSmoothTension（设置方法）

设置使用线性绘制时的曲线平滑参数。

```
void  LinesSmoothTension(
   const double  tension      // 参数值
   )
```

参数

*tension*

[in] 平滑参数值。

注意

'张力'值的范围是(0.0; 1.0]。

© 2000-2025, MetaQuotes Ltd.

## LinesSmoothStep（获得方法）

返回通过线性绘制时近似平滑线的长度。

```
double  LinesSmoothStep()
```

返回值

以像素为单位的近似直线的长度。

## LinesSmoothStep（设置方法）

设置通过线性绘制时近似平滑线的长度。

```
void  LinesSmoothStep(
   const double  step      // 线性长度
   )
```

参数

*step*

[in] 近似直线的长度

© 2000-2025, MetaQuotes Ltd.

## LinesEndStyle(设置方法)

获得使用线型模式绘制曲线时指明线型末端绘制样式的标识。

```
ENUM_LINE_END  LinesEndStyle()
```

返回值

使用线型模式绘制曲线时指明线型末端绘制样式的标识值。

## LinesEndStyle (获得方法)

设置使用线型模式绘制曲线时指明线型末端绘制样式的标识。

```
void  LinesEndStyle(
   ENUM_LINE_END  end_style      //标识值
   )
```

参数

*end_style*

[in] 使用线型模式绘制曲线时指明线型末端绘制样式的标识值。

© 2000-2025, MetaQuotes Ltd.

## LinesWidth（获得方法）

获得使用线型绘制曲线时的线型宽度。

```
int  LinesWidth()
```

返回值

线型宽度。

## LinesWidth（设置方法）

设置使用线型绘制曲线时的线型宽度。

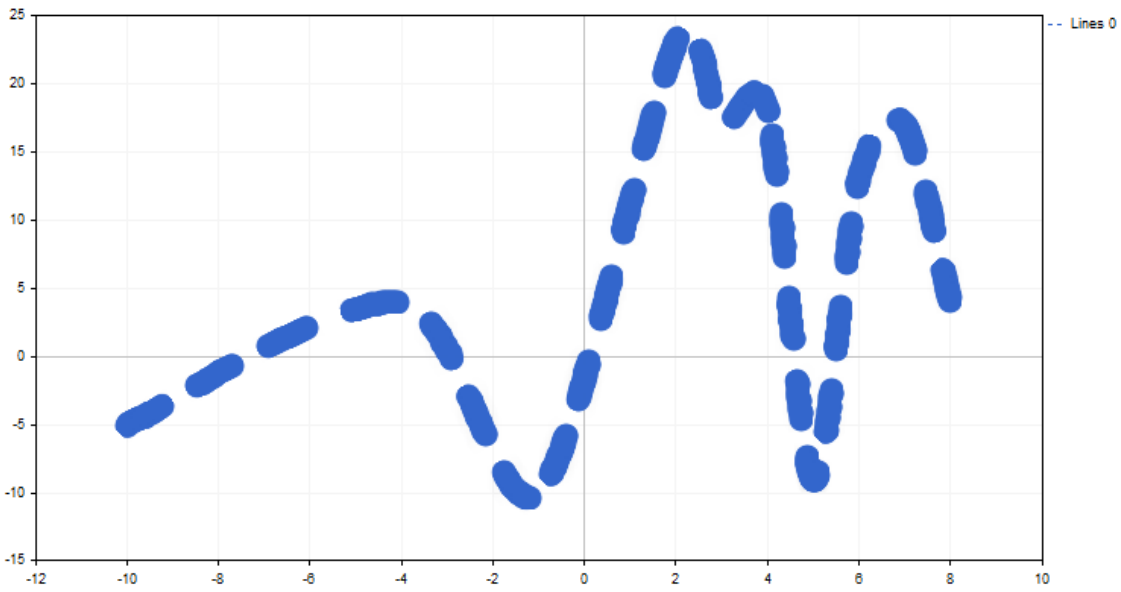
```
void  LinesWidth(
   const int  width      //线型宽度
   )
```

参数

*width*

[in] 使用线型绘制曲线时的线型宽度。

示例：




通过以下代码改变线型宽度：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                CandleGraphic.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   double x[]= { -100,-40,-10,20,30,40,50,60,70,80,120 };
   double y[]= { -5,4,-10,23,17,18,-9,13,17,4,9 };
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"ThickLineGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"ThickLineGraphic");
     }
//--- 创建曲线
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_LINES);
//--- 设置曲线属性
   curve.LinesSmooth(true);
   curve.LinesStyle(STYLE_DASH);
   curve.LinesEndStyle(LINE_END_ROUND);
   curve.LinesWidth(10);
//--- 绘制
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

## Histogram Width（获得方法）

返回使用直方图模式绘制时的列宽。

```
int  HistogramWidth()
```

返回值

以像素为单位的列宽。

## Histogram Width（设置方法）

设置使用直方图模式绘制时的列宽。

```
void  HistogramWidth(
   const int  width      //列宽
   )
```

参数

*宽度*

[in] 以像素为单位的列宽。

© 2000-2025, MetaQuotes Ltd.

## CustomPlotCBData（获得方法）

获得自定义曲线绘制模式中使用的对象指针。

```
void*  CustomPlotCBData()
```

返回值

用于自定义曲线绘制模式的对象指针。

## CustomPlotCBData（设置方法）

设置自定义曲线绘制模式中使用的对象指针。

```
void  CustomPlotCBData(
   void*  cbdata      // 对象指针
   )
```

参数

*cbdata*

[in] 自定义曲线绘制模式中使用的对象指针

© 2000-2025, MetaQuotes Ltd.

## Custom PlotFunction（获得方法）

获得实施自定义曲线绘制模式的函数指针。

```
PlotFucntion  CustomPlotFunction()
```

返回值

实施自定义曲线绘制模式的函数指针。

## Custom PlotFunction（设置方法）

设置实施自定义曲线绘制模式的函数指针。

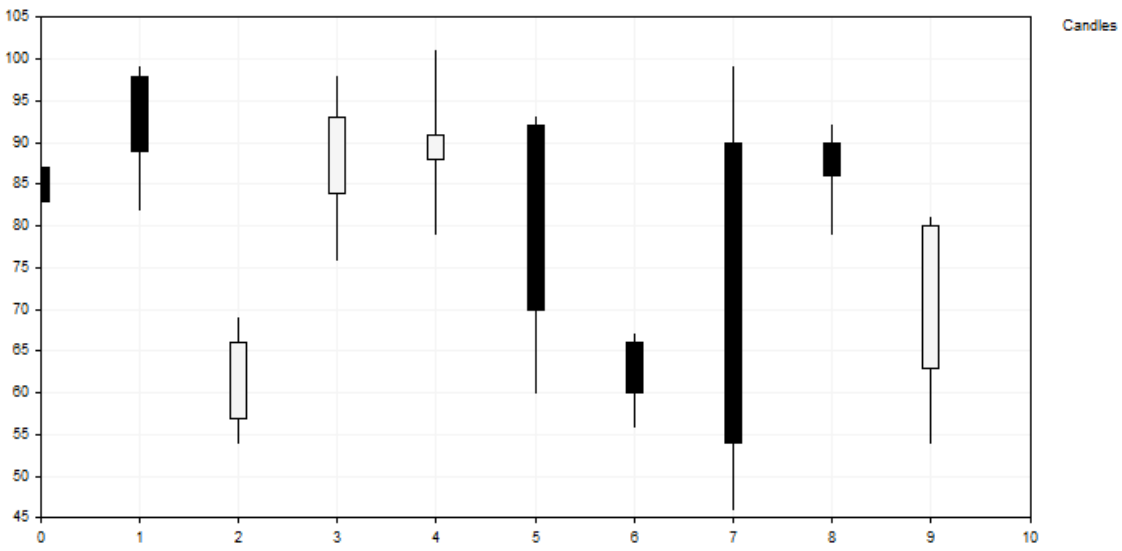
```
void  CustomPlotFunction(
   PlotFucntion  func      // 函数指针
   )
```

参数

*func*

[in] 实施自定义曲线绘制模式的函数指针

示例：




组成该曲线的柱形图通过以下代码来构建：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                                CandleGraphic.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//|CCandle 类                                                        |
//| 用法:表示蜡烛图的类                                             |
//+------------------------------------------------------------------+
class CCandle: public CObject
  {
private:
   double            m_open;
   double            m_close;
   double            m_high;
   double            m_low;
   uint              m_clr_inc;
   uint              m_clr_dec;
   int               m_width;

public:
                     CCandle(const double open,const double close,const double high,co
                                                    const int width,const uint clr_
                    ~CCandle(void);
   double            OpenValue(void)            const { return(m_open);      }
   double            CloseValue(void)           const { return(m_close);     }
   double            HigthValue(void)           const { return(m_high);      }
   double            LowValue(void)             const { return(m_low);       }
   uint              CandleColorIncrement(void) const { return(m_clr_inc);   }
   uint              CandleColorDecrement(void) const { return(m_clr_dec);   }
   int               CandleWidth(void)          const { return(m_width);     }
  };
//+------------------------------------------------------------------+
//| 构造函数                                                         |
//+------------------------------------------------------------------+
CCandle::CCandle(const double open,const double close,const double high,const double l
                                 const int width,const uint clr_inc=0x000000,const uin
                                 m_open(open),m_close(close),m_high(high),m_low(low),
                                 m_clr_inc(clr_inc),m_clr_dec(clr_dec),m_width(width)
  {
  }
//+------------------------------------------------------------------+
//| 析构函数                                                         |
//+------------------------------------------------------------------+
CCandle::~CCandle(void)
  {
  }
//+------------------------------------------------------------------+
//| 绘制蜡烛图的自定义方法                                           |
//+------------------------------------------------------------------+
void PlotCandles(double &x[],double &y[],int size,CGraphic *graphic,CCanvas *canvas,vo
  {
//--- 检查 obj
   CArrayObj *candles=dynamic_cast<CArrayObj*>(cbdata);
   if(candles==NULL || candles.Total()!=size)
      return;
//--- plot candles
   for(int i=0; i<size; i++)
     {
      CCandle *candle=dynamic_cast<CCandle*>(candles.At(i));
```

© 2000-2025, MetaQuotes Ltd.

```
      if(candle==NULL)
         return;
      //--- 主要计算
      int xc=graphic.ScaleX(x[i]);
      int width_2=candle.CandleWidth()/2;
      int open=graphic.ScaleY(candle.OpenValue());
      int close=graphic.ScaleY(candle.CloseValue());
      int high=graphic.ScaleY(candle.HigthValue());
      int low=graphic.ScaleY(candle.LowValue());
      uint clr=(open<=close) ?candle.CandleColorIncrement() :  candle.CandleColorDecre
      //--- plot candle
      canvas.LineVertical(xc,high,low,0x000000);
      //--- 绘制实体蜡烛图
      canvas.FillRectangle(xc+width_2,open,xc-width_2,close,clr);
      canvas.Rectangle(xc+width_2,open,xc-width_2,close,0x000000);
     }
  }
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   int count=10;
   int width=10;
   double x[];
   double y[];
   ArrayResize(x,count);
   ArrayResize(y,count);
   CArrayObj candles();
   double max=0;
   double min=0;
//--- 创建值
   for(int i=0; i<count; i++)
     {
      x[i] = i;
      y[i] = i;
      //--- 计算值
      double open=MathRound(50.0+(MathRand()/32767.0)*50.0);
      double close=MathRound(50.0+(MathRand()/32767.0)*50.0);
      double high=MathRound(MathMax(open,close)+(MathRand()/32767.0)*10.0);
      double low=MathRound(MathMin(open,close) -(MathRand()/32767.0)*10.0);
      //--- 寻找最大最小值
      if(i==0 || max<high)
         max=high;
      if(i==0 || min>low)
         min=low;
      //--- create candle
      CCandle *candle=new CCandle(open,close,high,low,width);
      candles.Add(candle);
     }
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"CandleGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"CandleGraphic");
     }
//--- 创建曲线
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_CUSTOM,"Candles");
//--- 设置曲线属性
   curve.CustomPlotFunction(PlotCandles);
   curve.CustomPlotCBData(GetPointer(candles));
```

© 2000-2025, MetaQuotes Ltd.

```
//--- 设置图形属性
   graphic.YAxis().Max((int)max);
   graphic.YAxis().Min((int)min);
//--- 绘制
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

## PointsType（获得方法）

获得绘制点式曲线时指明所用点式类型的标识。

```
ENUM_POINT_TYPE  PointsType()
```

返回值

指明点式类型的标识值。

## PointsType（设置方法）

设置绘制点式曲线时指明所用点式类型的标识。

```
void  PointsType(
   ENUM_POINT_TYPE  type      // 标识值
   )
```

参数

*type*

[in] 绘制点式曲线时指明所用点式类型的标识值。

© 2000-2025, MetaQuotes Ltd.

## StepsDimension（获得方法）

获得指明步进式曲线渲染所使用的维度的值。

```
int  StepsDimension()
```

返回值

步进式曲线渲染所使用的维度。

## StepsDimension（设置方法）

设置指明步进式曲线渲染所使用的维度的值。

```
void  StepsDimension(
   const int  dimension      // 维度
   )
```

参数

*dimension*

[in] 维度（0 或 1）。

注意

0 — x（垂直线后的水平线）。

1 — y（水平线后的垂直线）。

© 2000-2025, MetaQuotes Ltd.

## TrendLineCoefficients（获得方法）

获得写入数组的趋势线比率。

```
double&  TrendLineCoefficients()
```

返回值

趋势线比率。

## TrendLineCoefficients（设置方法）

设置写入数组的趋势线比率。

```
void  TrendLineCoefficients(
   double&  coefficients[]      // 编写比率的数组
   )
```

参数

*coefficients[]*

[out] 编写比率的数组。

© 2000-2025, MetaQuotes Ltd.

## TrendLineColor (获得方法)

获得曲线趋势线的颜色。

```
uint  TrendLineColor()
```

返回值

趋势线的颜色。

## TrendLineColor (设置方法)

设置曲线趋势线的颜色。

```
void  TrendLineColor(
   const uint  clr      // 趋势线的颜色
   )
```

参数

*clr*

[in] 线的颜色

© 2000-2025, MetaQuotes Ltd.

## TrendLineVisible（获得方法）

获得趋势线可视标识。

```
bool  TrendLineVisible()
```

返回值

指定趋势线是否可见的标识值。

## TrendLineVisible（设置方法）

设置趋势线可视标识。

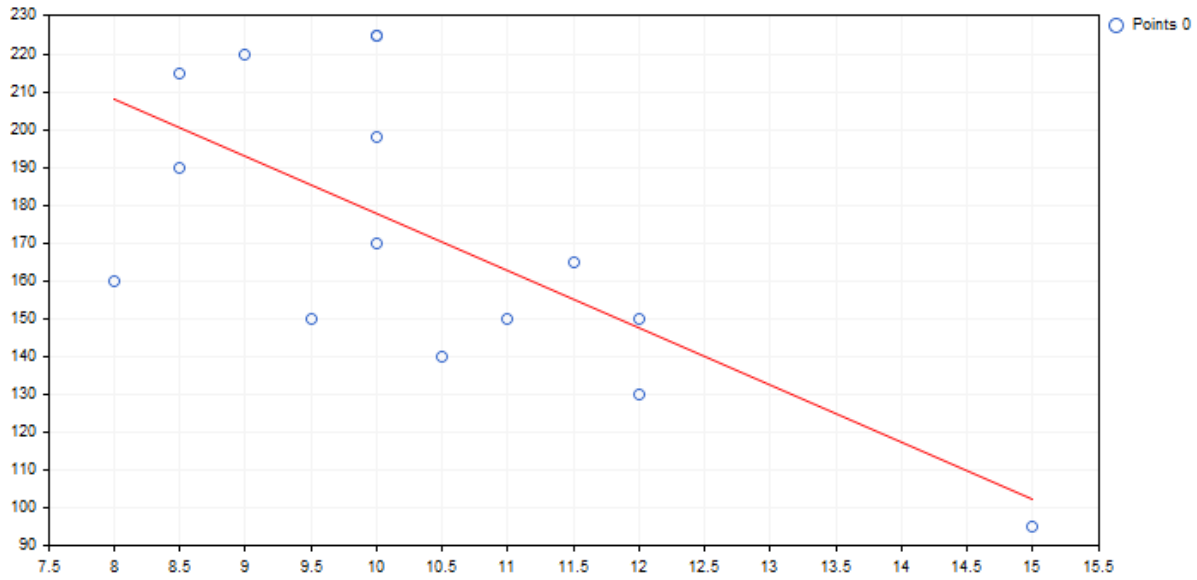
```
void  TrendLineVisible(
   const bool  visible      //标识值
   )
```

参数

*visible*

[in] 趋势线可视标识值。

示例：




以下即是提到的趋势线及其在图表绘制的代码：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                              TrendLineGraphic.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//| 脚本程序起始函数                                                 |
//+------------------------------------------------------------------+
void OnStart()
  {
   double x[]={12.0,11.5,11.0,12.0,10.5,10.0,9.0,8.5,10.0,8.5,10.0,8.0,9.5,10.0,15.0};
   double y[]={130.0,165.0,150.0,150.0,140.0,198.0,220.0,215.0,225.0,190.0,170.0,160.0
//--- 创建图形
   CGraphic graphic;
   if(!graphic.Create(0,"TrendLineGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"TrendLineGraphic");
     }
//--- 创建曲线
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_POINTS);
//--- 设置曲线属性
   curve.TrendLineVisible(true);
   curve.TrendLineColor(ColorToARGB(clrRed));
//--- 绘制
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

© 2000-2025, MetaQuotes Ltd.

# Update

更新曲线坐标。

Y坐标的工作版本。通过的数组指数在这里用作X坐标。

```
void  Update(
   const double&  y[]   // Y 坐标
   )
```

该版本使用X和Y坐标。

```
void  Update(
   const double&  x[],     // X 坐标
   const double&  y[]      // Y 坐标
   )
```

处理CPoint2D点的版本。

```
void  Update(
   const CPoint2D&  points[]      // 曲线坐标
   )
```

可以处理CurveFunction函数指针的版本。

```
void  Update(
   CurveFunction  function,     // 描述曲线的函数指针
   const double   from,         // 函数参数的初始值
   const double   to,           // 函数参数的最终值
   const double   step          // 参数增量
   )
```

参数

*x[]*

[in] X 坐标。

*y[]*

[in] Y 坐标。

*points[]*

[in] 曲线坐标。

*function*

[in]描述曲线的函数指针

*from*

[in] 函数参数的初始值

*to*

[in] 函数参数的最终值

*step*

[in] 参数增量

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

## Visible (获得方法)

获得定义函数在图表是否可见的标识。

```
void  Visible(
   const bool  visible      //
   )
```

返回值

定义函数在图表可见的标识值。

## Visible (设置方法)

设置定义函数在图表是否可见的标识。

```
void  Visible(
   const bool  visible      // 标识值
   )
```

参数

*visible*

[in] 定义函数在图表可见的标识值。

© 2000-2025, MetaQuotes Ltd.

# CGraphic

CGraphic是创建自定义图表的基本类。

## 描述

CGraphic类提供了使用自定义图表的众多方面。

该类存储主要的图表元素，设置其参数和执行绘图。

另外，该类还存储图表曲线和提供不同的显示选项。

## 声明

```
class CGraphic
```

## 主题

```
#include <Graphics\Graphic.mqh>
```

## 类函数

| Method | 描述 |
| --- | --- |
| Create | 创建一个绑定图表对象的图形源 |
| Destroy | 删除图表并取消图形资源 |
| Update | 显示实施的更改 |
| ChartObjectName | 获得绑定图表的对象名称 |
| ResourceName | 获得图形资源名称 |
| XAxis | 获得X坐标轴的指针 |
| YAxis | 获得Y坐标轴的指针 |
| GapSize | 获得/设置图表元素之间的缩进大小 |
| BackgroundColor | 获得/设置背景色 |
| BackgroundMain | 获得/设置图表标题 |
| BackgroundMainSize | 获得/设置子标题字体大小 |
| BackgroundMainColor | 获得/设置图表标题颜色 |
| BackgroundSub | 获得/设置子标题 |
| BackgroundSubSize | 获得/设置子标题字体大小 |
| BackgroundSubColor | 获得/设置图表子标题颜色 |
| GridLineColor | 获得/设置网格线颜色 |
| GridBackgroundColor | 获得/设置网格背景色 |

© 2000-2025, MetaQuotes Ltd.

| Method | 描述 |
| --- | --- |
| GridCircleRadius | 获得/设置网格节点的点半径 |
| GridCircleColor | 获得/设置网格节点的点颜色 |
| GridHasCircle | 获得/设置网格节点的点式绘制标识 |
| GridAxisLineColor | 获得真实图表坐标轴的颜色值。 |
| HistoryNameWidth | 获得/设置可允许显示的最长曲线名 |
| HistoryNameSize | 获得/设置曲线名称的字体大小 |
| HistorySymbolSize | 获得/设置标记法则交易品种的大小 |
| TextAdd | 添加文本到图表 |
| LineAdd | 添加线型到图表 |
| CurveAdd | 创建和添加曲线到图表 |
| CurvePlot | 通过索引绘制之前创建的曲线 |
| CurvePlotAll | 绘制之前创建的全部曲线 |
| CurveGetByIndex | 根据指定索引获得曲线 |
| CurveGetByName | 通过指定名称获得曲线 |
| CurveRemoveByIndex | 根据指定索引删除曲线。 |
| CurveRemoveByName | 通过指定名称删除曲线。 |
| CurvesTotal | 获得给定图表的曲线数。 |
| MarksToAxisAdd | 添加刻度标记到图表坐标轴 |
| MajorMarkSize | 获得/设置图表坐标轴上的报价刻度大小 |
| FontSet | 设置当前字体参数 |
| FontGet | 获得当前字体参数 |
| Attach | 获得/设置图形资源并将其绑定到CGraphic类实例 |
| CalculateMaxMinValues | 计算（重新计算）两个坐标轴上的最大最小图表值。 |
| Height | 获得图表像素高度。 |
| IndentDown | 获得/设置较低边界的图表缩进。 |
| IndentLeft | 获得/设置左边界的图表缩进。 |
| IndentRight | 获得/设置右边界的图表缩进。 |
| IndentUp | 获得/设置上边界的图表缩进。 |
| Redraw | 重新绘制图表。 |

© 2000-2025, MetaQuotes Ltd.

| Method | 描述 |
| --- | --- |
| ResetParameters | 重置图表重绘参数。 |
| ScaleX | 按比例衡量X坐标轴的值。 |
| ScaleY | 按比例衡量Y坐标轴的值。 |
| SetDefaultParameters | 将图表参数设为默认值。 |
| Width | 获得图表像素宽度。 |

© 2000-2025, MetaQuotes Ltd.

# 创建

创建 绑定图表对象的图形资源。

```
bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 子窗口索引
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y1 坐标
   )
```

参数

*chart*

[in] 图表ID。

*name*

[in] 名称。

*subwin*

[in] 子窗口索引。

*x1*

[in] X1 坐标。

*y1*

[in] Y1 坐标。

*x2*

[in] X2 坐标。

*y2*

[in] Y2 坐标。

© 2000-2025, MetaQuotes Ltd.

# Destroy

移除图表并破坏图形资源。

```
void  Destroy()
```

© 2000-2025, MetaQuotes Ltd.

# Update

显示实施的更改。

```
void  Update(
   const bool  redraw=true      //标识
   )
```

参数

*redraw=true*

[in] 标识值

© 2000-2025, MetaQuotes Ltd.

# ChartObjectName

获取绑定图表的对象名称。

```
string  ChartObjectName()
```

返回值

绑定图表的对象名称。

© 2000-2025, MetaQuotes Ltd.

# ResourceName

接收图形资源名称。

```
string  ResourceName()
```

返回值

图形资源名称。

© 2000-2025, MetaQuotes Ltd.

# XAxis

返回X坐标轴的指针。

```
CAxis  *XAxis()
```

返回值

X坐标轴的指针。

© 2000-2025, MetaQuotes Ltd.

# YAxis

返回Y坐标轴的指针。

```
CAxis  *YAxis()
```

返回值

Y坐标轴的指针。

© 2000-2025, MetaQuotes Ltd.

## GapSize（获得方法）

返回图表要素间的缩进尺寸。

```
int  GapSize()
```

返回值

以像素为单位的缩进尺寸。

## GapSize（设置方法）

设置图表要素间的缩进尺寸。

```
void  GapSize(
   const int  size      // 缩进尺寸
   )
```

参数

*size*

[in] 以像素为单位的缩进尺寸。

© 2000-2025, MetaQuotes Ltd.

## BackgroundColor (获得方法)

返回背景色。

```
color  BackgroundColor()
```

## BackgroundColor (设置方法)

设置背景色。

```
void  BackgroundColor(
   const color  clr      // 背景色
   )
```

参数

*clr*

[in] 背景色。

© 2000-2025, MetaQuotes Ltd.

## BackgroundMain（获得方法）

返回图表标题

```
string  BackgroundMain()
```

## BackgroundMain（设置方法）

设置图表标题文本。

```
void  BackgroundMain(
   const string  main      // 标题文本
   )
```

参数

*main*

[in] 图表标题文本

© 2000-2025, MetaQuotes Ltd.

## BackgroundMainSize（获得方法）

返回标题大小。

```
int  BackgroundMainSize()
```

返回值

标题字体大小。

## BackgroundMainSize（设置方法）

设置标题大小。

```
void  BackgroundMainSize(
   const int  size     // 标题大小
   )
```

参数

*size*

[in] 标题字体大小。

© 2000-2025, MetaQuotes Ltd.

## BackgroundMainColor (获得方法)

返回标题颜色。

```
color  BackgroundMainColor()
```

返回值

标题颜色。

## BackgroundMainColor (设置方法)

设置标题颜色。

```
void  BackgroundMainColor(
   const color  clr      // 标题颜色
   )
```

参数

*clr*

[in] 标题颜色。

© 2000-2025, MetaQuotes Ltd.

## BackgroundSub (获得方法)

返回副标题。

```
string  BackgroundSub()
```

返回值

副标题文本。

## BackgroundSub (设置方法)

设置副标题文本。

```
void  BackgroundSub (Set method)(
   const string  sub       // 副标题文本
   )
```

参数

*sub*

[in] 副标题文本。

© 2000-2025, MetaQuotes Ltd.

## BackgroundSubSize（获得方法）

返回副标题字体大小

```
int BackgroundSubSize()
```

## BackgroundSubSize（获得方法）

设置副标题大小

```
void  BackgroundSubSize(
   const int  size      // 副标题字体大小
   )
```

参数

*size*

[in] 副标题字体大小

© 2000-2025, MetaQuotes Ltd.

## BackgroundSubColor (获得方法)

返回副标题颜色。

```
color  BackgroundSubColor()
```

## BackgroundSubColor (设置方法)

设置副标题颜色。

```
void  BackgroundSubColor(
   const color  clr      // 副标题颜色
   )
```

参数

*clr*

[in] 副标题颜色。

© 2000-2025, MetaQuotes Ltd.

## GridLineColor (获得方法)

返回网格线颜色

```
color  GridLineColor()
```

返回值

网格线颜色。

## GridLineColor (设置方法)

设置网格线颜色。

```
void  GridLineColor(
   const color  clr      // 线颜色
   )
```

参数

*clr*

[in] 网格线颜色。

© 2000-2025, MetaQuotes Ltd.

## GridBackgroundColor (获得方法)

返回网格背景色

```
color  GridBackgroundColor()
```

返回值

网格背景色

## GridBackgroundColor (设置方法)

设置网格背景色

```
void  GridBackgroundColor(
   const color  clr      // 网格背景色
   )
```

参数

*clr*

[in] 网格背景色

© 2000-2025, MetaQuotes Ltd.

## GridCircleRadius（获得方法）

返回网络节点中的点半径。

```
int  GridCircleRadius()
```

返回值

以像素为单位的点半径。

## GridCircleRadius（设置方法）

设置网络节点中的点半径

```
void  GridCircleRadius(
   const int  r      // 半径
   )
```

参数

*r*

[in] 以像素为单位的点半径。

© 2000-2025, MetaQuotes Ltd.

## GridCircleColor (获得方法)

返回网络节点中的点颜色。

```
color  GridCircleColor()
```

返回值

点颜色。

## GridCircleColor (设置方法)

设置网络节点中的点颜色。

```
void  GridCircleColor(
   const color  clr      // 点颜色
   )
```

参数

*clr*

[in] 点颜色。

© 2000-2025, MetaQuotes Ltd.

## GridHasCircle（获得方法）

返回定义是否应该显示网格节点中的点的标识。

```
bool  GridHasCircle()
```

返回值

标识值。

注意

true —显示点

false —不显示点

## GridHasCircle（设置方法）

设置定义是否应该显示网格节点中的点的标识。

```
void  GridHasCircle(
   const bool  has
   )
```

参数

*has*

[in]标识值。

注意

true —显示点

false —不显示点

© 2000-2025, MetaQuotes Ltd.

## GridAxisLineColor (获得方法)

获得真实图表坐标轴的颜色值。

```
uint  GridAxisLineColor()
```

返回值

真实图表坐标轴的颜色。

## GridAxisLineColor (设置方法)

设置真实图表坐标轴的颜色值。

```
void  GridAxisLineColor(
   const uint  clr      // 图表坐标轴的颜色
   )
```

参数

*clr*

[in] 真实图表坐标轴的颜色。

© 2000-2025, MetaQuotes Ltd.

## HistoryNameWidth（获得方法）

返回可允许显示的最长曲线名称。

```
int  HistoryNameWidth()
```

返回值

以像素为单位的最大长度。

注意

如果曲线名称超出了允许的最大长度，则会将其缩短并加缩写点结束。

## HistoryNameWidth（设置方法）

设置可允许显示的最长曲线名称。

```
void  HistoryNameWidth(
   const int  width      // 最大长度
   )
```

参数

*宽度*

[in] 以像素为单位的最大长度。

注意

如果曲线名称超出了允许的最大长度，则会将其缩短并加缩写点结束。

© 2000-2025, MetaQuotes Ltd.

## HistoryNameSize（获得方法）

返回曲线名称的字体大小。

```
int  HistoryNameSize()
```

返回值

曲线名称的字体大小。

## HistoryNameSize（设置方法）

设置曲线名称的字体大小。

```
void  HistoryNameSize (Set method)(
   const int  size      // 名称字体大小
   )
```

参数

*size*

[in] 名称字体大小。

© 2000-2025, MetaQuotes Ltd.

## HistorySymbolSize（获得方法）

返回图表符号约定交易品种的大小

```
int  HistorySymbolSize()
```

返回值

符号约定交易品种的大小

## HistorySymbolSize（设置方法）

设置图表符号约定交易品种的大小

```
void  HistorySymbolSize(
   const int  size      // 交易品种大小
   )
```

参数

*size*

[in] 符号约定交易品种的大小。

© 2000-2025, MetaQuotes Ltd.

# TextAdd

将文本添加到图表。

使用X和Y坐标的版本

```
void  TextAdd(
   const int      x,                // X 坐标
   const int      y,                // Y 坐标
   const string   text,             // 文本
   const uint     clr,              // 颜色
   const uint     alignment=0       // 对齐
   )
```

CPoint版本

```
void  TextAdd(
   const CPoint   &point,           // 点坐标
   const string   text,             // 文本
   const uint     clr,              // 颜色
   const uint     alignment=0       // 对齐
   )
```

参数

*x*

[in] X坐标。

*y*

[in] Y坐标。

*&point*

[in] 点坐标。

*text*

[in] 文本。

*clr*

[in] 颜色。

*alignment=0*

[in] 对齐。

© 2000-2025, MetaQuotes Ltd.

# LineAdd

将直线添加到图表。

该版本使用X和Y坐标

```
void  LineAdd(
   const int   x1,         // x1 坐标
   const int   y1,         // y1 坐标
   const int   x2,         // x2 坐标
   const int   y2,         // y2 坐标
   const uint  clr,        // 颜色
   const uint  style       // 风格
   )
```

CPoint版本

```
void  LineAdd2(
   const CPoint  &point1,     // 第一个点坐标
   const CPoint  &point2,     // 第二个点坐标
   const uint    clr,         // 颜色
   const uint    style        // 风格
   )
```

参数

*x1*

[in] X1 坐标。

*y1*

[in] Y1 坐标。

*x2*

[in] X2 坐标。

*y2*

[in] Y2 坐标。

*&point1*

[in] 第一个点坐标。

*&point2*

[in] 第二个点坐标。

*clr*

[in] 颜色。

*style*

[in] 风格。

© 2000-2025, MetaQuotes Ltd.

# CurveAdd

创建并添加新曲线到图表。

该版本使用Y坐标（自动设定曲线颜色）

```
CCurve* CurveAdd(
   const double      &y[],          // Y 坐标
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

注意

Y数组索引用作曲线的X坐标。

该版本使用X和Y坐标（自动设定曲线颜色）

```
CCurve*  CurveAdd(
   const double      &x[],          // X 坐标
   const double      &y[],          // Y 坐标
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

该版本使用CPoint2D点（自动设定曲线颜色）

```
CCurve*  CurveAdd(
   const CPoint2D    &points[],     // 点坐标
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

该版本使用CurveFunction函数指针（自动设定曲线颜色）

```
CCurve*  CurveAdd(
   CurveFunction     function,      // 函数指针
   const double      from,          // 参数初始值
   const double      to,            // 参数最终值
   const double      step,          // 参数盈余
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

该版本使用Y坐标（用户设定曲线颜色）

© 2000-2025, MetaQuotes Ltd.

```
CCurve*  CurveAdd(
   const double      &y[],          // Y 坐标
   const uint        clr,           // 曲线颜色
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

注意

Y数组索引用作曲线的X坐标。

该版本使用X和Y坐标（用户设定曲线颜色）

```
CCurve*  CurveAdd(
   const double      &x[],          // X 坐标
   const double      &y[],          // Y 坐标
   const uint        clr,           // 曲线颜色
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

该版本使用CPoint2D点（用户设定曲线颜色）

```
CCurve*  CurveAdd(
   const CPoint2D    &points[],     // 点坐标
   const uint        clr,           // 曲线颜色
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

该版本使用CurveFunction函数指针（用户设定曲线颜色）

```
CCurve*  CurveAdd(
   CurveFunction     function,      // 函数指针
   const double      from,          // 参数初始值
   const double      to,            // 参数最终值
   const double      step,          // 参数盈余
   const uint        clr,           // 曲线颜色
   ENUM_CURVE_TYPE   type,          // 曲线类型
   const string      name=NULL      // 曲线名称
   )
```

参数

*&x[]*

© 2000-2025, MetaQuotes Ltd.

[in] X 坐标。

*&y[]*

[in] Y 坐标。

*&points[]*

[in] 点坐标。

*function*

[in] 函数指针。

*from*

[in] 参数初始值。

*to*

[in] 参数最终值。

*step*

[in] 参数盈余。

*type*

[in] 曲线类型。

*name=NULL*

[in] 曲线名称。

*clr*

[in] 曲线颜色。

### 返回值

已创建曲线的指针。

© 2000-2025, MetaQuotes Ltd.

# CurvePlot

通过索引显示之前创建的曲线。

```
bool  CurvePlot(
   const int  index      // 索引
   )
```

参数

*index*

[in] 曲线索引

返回值

true - 成功，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# CurvePlotAll

显示之前添加到图表的全部曲线。

```
bool  CurvePlotAll()
```

返回值

true - 成功，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# CurveGetByIndex

获取指定索引的曲线。

```
CCurve*  CurveGetByIndex(
   const int  index      // 曲线索引
   )
```

参数

*index*

[in] 曲线索引。

返回值

指定索引的曲线指针。

© 2000-2025, MetaQuotes Ltd.

# CurveGetByName

获取指定名称的曲线。

```
CCurve*  CurveGetByName(
   const string  name      // 曲线名称
   )
```

参数

*name*

[in] 曲线名称。

返回值

第一个找到的有指定名称的曲线指针。

© 2000-2025, MetaQuotes Ltd.

# CurveRemoveByIndex

根据指定索引删除曲线。

```
bool  CurveRemoveByIndex(
   const int  index      // 曲线索引
   )
```

参数

*index*

[in] 将被删除的曲线索引。

返回值

true —成功，否则 —false。

© 2000-2025, MetaQuotes Ltd.

# CurveRemoveByName

通过指定名称删除曲线。

```
bool  CurveRemoveByName(
   const string  name      // 曲线名称
   )
```

参数

*name*

[in] 将被删除的曲线名称。

返回值

true —成功，否则 —false。

© 2000-2025, MetaQuotes Ltd.

# CurvesTotal

获得给定图表的曲线数。

```
int  CurvesTotal()
```

返回值

曲线数量。

注意

无论何种绘制和可视样式，都会考虑当前图表的全部曲线。

© 2000-2025, MetaQuotes Ltd.

# MarksToAxisAdd

将比例标记（报价）添加到图表坐标轴。

```
bool  MarksToAxisAdd(
   const double         &marks[],        // 报价坐标
   const int            mark_size,       // 最小报价单位
   ENUM_MARK_POSITION   position,        // 报价位置
   const int            dimension=0      // 尺寸
   )
```

参数

*&marks[]*

[in] 报价坐标

*mark_size*

[in] 最小报价单位

*position*

[in] 报价位置

*dimension=0*

[in] 0 — 添加到 X 坐标轴，

1 — 添加到 Y 坐标轴

返回值

true - 成功，否则 - false。

© 2000-2025, MetaQuotes Ltd.

## MajorMarkSize（获得方法）

返回坐标轴上报价比例单位。

```
int  MajorMarkSize()
```

## MajorMarkSize（设置方法）

返回坐标轴上报价比例单位。

```
void  MajorMarkSize(
   const int  size      // 最小报价单位
   )
```

参数

*size*

[in] 以像素为单位的最小报价量。

© 2000-2025, MetaQuotes Ltd.

# FontSet

设置当前字体参数。

```
bool  FontSet(
   const string  name,          // 名称
   const int     size,          // 大小
   const uint    flags=0,       // 标识
   const uint    angle=0        // 角度
   )
```

参数

*name*

[in] 名称。

*size*

[in] 大小。

*flags=0*

[in] 标识。

*angle=0*

[in] 角度。

返回值

true - 成功，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# FontGet

获取当前字体参数。

```
void  FontGet(
   string  &name,      // 名称
   int     &size,      // 大小
   uint    &flags,     //标识
   uint    &angle      // 角度
   )
```

参数

*&name*

[out] 名称。

*&size*

[out] 大小。

*&flags*

[out] 标识。

*&angle*

[out] 角度。

© 2000-2025, MetaQuotes Ltd.

# Attach

从OBJ_BITMAP_LABEL对象获得图形资源并将其绑定到CGraphic类实例的版本：

```
bool  Attach(
   const long     chart_id,      // 图表ID
   const string   objname        // 图形对象名称
   )
```

创建OBJ_BITMAP_LABEL对象图形资源和将其绑定到CGraphic类实例的版本：

```
bool  Attach(
   const long     chart_id,      // 图表ID
   const string   objname,       // 图形对象名称
   const int      width,         // 图像宽度
   const int      height         // 图像高度
   )
```

参数

*chart_id*

[in] 图表 ID。

*objname*

[in] 图形对象的名称。

*width*

[in] 资源中的图像宽度。

*height*

[in] 资源中的图像高度。

返回值

true —成功，false —绑定对象失败。

© 2000-2025, MetaQuotes Ltd.

# CalculateMaxMinValues

计算（重新计算）两个坐标轴上的最大最小图表值。

```
void  CalculateMaxMinValues()
```

© 2000-2025, MetaQuotes Ltd.

# Height

获得图表像素高度。

```
int  Height()
```

返回值

图表像素高度。

© 2000-2025, MetaQuotes Ltd.

## IndentDown（获得方法）

获得较低边界的图表缩进。

```
int  IndentDown()
```

返回值

缩进的像素大小。

## IndentDown（设置方法）

设置较低边界的图表缩进。

```
void  IndentDown(
   const int  down      // 缩进大小
   )
```

参数

*down*

[in]缩进的像素大小。

© 2000-2025, MetaQuotes Ltd.

## IndentLeft (获得方法)

获得左边界的图表缩进。

```
int  IndentLeft()
```

返回值

缩进的像素大小。

## IndentLeft (设置方法)

设置左边界的图表缩进。

```
void  IndentLeft(
   const int  left      // 缩进大小
   )
```

参数

*left*

[in]缩进的像素大小。

© 2000-2025, MetaQuotes Ltd.

## IndentRight (获得方法)

获得右边界的图表缩进。

```
int  IndentRight()
```

返回值

缩进的像素大小。

## IndentRight (设置方法)

设置右边界的图表缩进。

```
void  IndentRight(
   const int  right      // 缩进大小
   )
```

参数

*right*

[in] 缩进的像素大小。

© 2000-2025, MetaQuotes Ltd.

## IndentUp (获得方法)

获得上边界的图表缩进。

```
int  IndentUp()
```

返回值

缩进的像素大小。

## IndentUp (设置方法)

设置上边界的图表缩进。

```
void  IndentUp(
   const int  up      // 缩进大小
   )
```

参数

*up*

[in] 缩进的像素值。

© 2000-2025, MetaQuotes Ltd.

# Redraw

重新绘制图表。

```
bool  Redraw(
   const bool  rescale=false      // 标识值
   )
```

参数

*rescale=false*

[in] 指明图表是否应该重新调整比例的标识。

返回值

true - 成功，否则 — false。

© 2000-2025, MetaQuotes Ltd.

# ResetParameters

重置图表重绘参数。

```
void  ResetParameters()
```

© 2000-2025, MetaQuotes Ltd.

# ScaleX

按比例衡量X坐标轴的值。

```
virtual int  ScaleX(
   double  x      // X坐标轴的值
   )
```

参数

*x*

[in] X坐标轴的真实值。

返回值

像素值。

注意

按像素调整显示在图表上的真实值。

© 2000-2025, MetaQuotes Ltd.

# ScaleY

按比例衡量Y坐标轴的值。

```
virtual int  ScaleY(
   double  y      // Y坐标轴的值
   )
```

参数

*y*

[in] Y坐标轴的真实值。

返回值

像素值。

注意

按像素调整显示在图表上的真实值。

© 2000-2025, MetaQuotes Ltd.

# SetDefaultParameters

将图表参数设为默认值。

```
void  SetDefaultParameters()
```

© 2000-2025, MetaQuotes Ltd.

# Width

获得图表像素宽度。

```
int  Width()
```

返回值

图表像素宽度。

© 2000-2025, MetaQuotes Ltd.

# 技术指标和时间序列

这章节包含技术指标和时间序列类的技术细节，以及 MQL5 标准库中相关组件的描述。

使用技术指标和时间序列类来开发应用 (脚本，智能交易)，可节约大量时间。

MQL5 标准库位于终端工作目录的 Include\Indicators 文件夹。

| 类/组 | 描述 |
| --- | --- |
| 基准类 | 基准和辅助类组 |
| 时间序列类 | 时间序列类组 |
| 趋势指标 | 趋势指标类组 |
| 振荡器 | 振荡器类组 |
| 成交量指标 | 交易量指标类组 |
| 比尔•威廉姆斯指标 | 比尔•威廉姆斯指标类 |
| 自定义指标 | 自定义指标类 |

© 2000-2025, MetaQuotes Ltd.

# 基准与辅助技术指标类以及时间序列类

这章节包含基准与辅助技术指标和时间序列类的技术细节，以及 MQL5 标准库中相关组件的描述。

| 类/组 | 描述 |
| --- | --- |
| CSpreadBuffer | 历史点差缓存区类 |
| CTimeBuffer | 历史开盘价缓存区类 |
| CTickVolumeBuffer | 历史报价交易量缓存区类 |
| CRealVolumeBuffer | 历史实际交易量缓存区类 |
| CDoubleBuffer | 双精度数据缓存区基准类 |
| COpenBuffer | 柱线开盘价缓存区类 |
| CHighBuffer | 柱线最高价缓存区类 |
| CLowBuffer | 柱线最低价缓存区类 |
| CCloseBuffer | 柱线收盘价缓存区类 |
| CIndicatorBuffer | 技术指标缓存区类 |
| CSeries | 访问时间序列数据的基准类 |
| CPriceSeries | 访问价格数据的基准类 |
| CIndicator | 技术指标基准类 |
| CIndicators | 技术指标和序列集合 |

方法继承自类 CArrayObj

FreeMode, FreeMode, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# CSpreadBuffer

CSpreadBuffer 类用于简便地访问历史柱线的点差。

## 描述

CSpreadBuffer 类可供访问历史柱线的点差。

## 声明

```
class CSpreadBuffer: public CArrayInt
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayInt
      CSpreadBuffer

## 类方法

| 属性 | |
|---|---|
| Size | 设置缓存区大小 |
| 设置 | |
| SetSymbolPeriod | 设置品名和周期 |
| 数据访问方法 | |
| At | 获取缓存区索引处的元素。 |
| 数据更新方法 | |
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayInt

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

© 2000-2025, MetaQuotes Ltd.

# Size

设置缓存区大小。

```
void  Size(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

设置品名和周期。

```
void  SetSymbolPeriod(
   const string           symbol,     // 品名
   const ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 新品名。

*period*

[输入] 新周期 (ENUM_TIMEFRAMES 枚举)。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
int  At(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CTimeBuffer

CTimeBuffer类用于简便地访问历史柱线的开盘时间。

## 描述

CTimeBuffer类可供访问历史柱线的开盘时间。

## 声明

```
class CTimeBuffer: public CArrayLong
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayLong
      CTimeBuffer

## 类方法

| 属性 | |
|---|---|
| Size | 设置缓存区大小 |
| 设置 | |
| SetSymbolPeriod | 设置品名和周期 |
| 数据访问方法 | |
| At | 获取缓存区索引处的元素。 |
| 数据更新方法 | |
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

© 2000-2025, MetaQuotes Ltd.

# Size

设置缓存区大小。

```
void  Size(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

设置品名和周期。

```
void  SetSymbolPeriod(
   const string           symbol,     // 品名
   const ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 新品名。

*period*

[输入] 新周期 (ENUM_TIMEFRAMES 枚举)。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
long  At(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CTickVolumeBuffer

CTickVolumeBuffer类用于简便地访问历史柱线的报价交易量。

## 描述

CTickVolumeBuffer类可供访问历史柱线的报价交易量。

## 声明

```
class CTickVolumeBuffer: public CArrayLong
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayLong
      CTickVolumeBuffer

## 类方法

| 属性 | |
|---|---|
| Size | 设置缓存区大小 |
| 设置 | |
| SetSymbolPeriod | 设置品名和周期 |
| 数据访问方法 | |
| At | 获取缓存区索引处的元素。 |
| 数据更新方法 | |
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

© 2000-2025, MetaQuotes Ltd.

## Size

设置缓存区大小。

```
void  Size(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

设置品名和周期。

```
void  SetSymbolPeriod(
   const string           symbol,     // 品名
   const ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 新品名。

*period*

[输入] 新周期 (ENUM_TIMEFRAMES 枚举)。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
long  At(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CRealVolumeBuffer

CRealVolumeBuffer类用于简便地访问历史柱线的实际交易量。

## 描述

CTickVolumeBuffer类可供访问历史柱线的实际交易量。

## 声明

```
class CRealVolumeBuffer: public CArrayLong
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayLong
      CRealVolumeBuffer

## 类方法

| 属性 | |
|---|---|
| Size | 设置缓存区大小 |
| 设置 | |
| SetSymbolPeriod | 设置品名和周期 |
| 数据访问方法 | |
| At | 获取缓存区索引处的元素。 |
| 数据更新方法 | |
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

© 2000-2025, MetaQuotes Ltd.

# Size

设置缓存区大小。

```
void  Size(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

设置品名和周期。

```
void  SetSymbolPeriod(
   const string           symbol,     // 品名
   const ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 新品名。

*period*

[输入] 新周期 (ENUM_TIMEFRAMES 枚举)。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
long  At(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CDoubleBuffer

CDoubleBuffer类用于简便地访问双精度型缓存区数据。

## 描述

CDoubleBuffer类可供简便地访问双精度型缓存区数据。

## 声明

```
class CDoubleBuffer: public CArrayDouble
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
CArray
CArrayDouble
CDoubleBuffer

直接分支

CCloseBuffer, CHighBuffer, CIndicatorBuffer, CLowBuffer, COpenBuffer

## 类方法

| 属性 | |
|---|---|
| Size | 设置缓存区大小 |
| 设置 | |
| SetSymbolPeriod | 设置品名和周期 |
| 数据访问方法 | |
| At | 获取缓存区数据 |
| 数据更新方法 | |
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayDouble

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

© 2000-2025, MetaQuotes Ltd.

## Size

设置缓存区大小。

```
void  Size(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# SetSymbolPeriod

设置品名和周期。

```
void  SetSymbolPeriod(
   const string           symbol,     // 品名
   const ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 新品名。

*period*

[输入] 新周期 (ENUM_TIMEFRAMES 枚举)。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
double  At(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前 (第零号) 缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# COpenBuffer

COpenBuffer类用于简便地访问历史柱线的开盘价。

## 描述

COpenBuffer类可供访问历史柱线的开盘价。

## 声明

```
class COpenBuffer: public CDoubleBuffer
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        COpenBuffer

## 类方法

| 数据更新方法 | |
|---|---|
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

方法继承自类CDoubleBuffer

Size, At, SetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前（第零号）缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CHighBuffer

CHighBuffer类用于简便地访问历史柱线的最高价。

## 描述

CHighBuffer类可供访问历史柱线的最高价。

## 声明

```
class CHighBuffer: public CDoubleBuffer
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CHighBuffer

## 类方法

| 数据更新方法 | |
|---|---|
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

方法继承自类CDoubleBuffer

Size, At, SetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前 (第零号) 缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CLowBuffer

CLowBuffer类用于简便地访问历史柱线的最低价。

## 描述

CLowBuffer类可供访问历史柱线的最低价。

## 声明

```
class CLowBuffer: public CDoubleBuffer
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CLowBuffer

## 类方法

| 数据更新方法 | |
|---|---|
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

方法继承自类CDoubleBuffer

Size, At, SetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前 (第零号) 缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CCloseBuffer

CCloseBuffer类用于简便地访问历史柱线的收盘价。

## 描述

CCloseBuffer类可供访问历史柱线的收盘价。

## 声明

```
class CCloseBuffer: public CDoubleBuffer
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
CArray
CArrayDouble
CDoubleBuffer
CCloseBuffer

## 类方法

| 数据更新方法 | |
|---|---|
| virtual Refresh | 更新缓存区 |
| virtual RefreshCurrent | 更新当前值 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

方法继承自类CDoubleBuffer

Size, At, SetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新缓存区。

```
virtual bool  Refresh()
```

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
virtual bool  RefreshCurrent()
```

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CIndicatorBuffer

CIndicatorBuffer 类用于简便地访问指标的缓存区数据。

## 描述

CIndicatorBuffer 类可供简便地访问技术指标的缓存区数据。

## 声明

```
class CIndicatorBuffer: public CDoubleBuffer
```

## 标称库文件

```
#include <Indicators\Indicator.mqh>
```

## 继承体系

CObject
CArray
CArrayDouble
CDoubleBuffer
CIndicatorBuffer

## 类方法

| 属性 | |
|---|---|
| Offset | 获取/设置缓存区位移 |
| 名称 | 获取/设置缓存区名称 |
| 数据访问方法 | |
| At | 获取缓存区中的元素 |
| 数据更新方法 | |
| Refresh | 更新缓存区 |
| RefreshCurrent | 仅更新当前值 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

方法继承自类CDoubleBuffer

Size, At, SetSymbolPeriod

© 2000-2025, MetaQuotes Ltd.

## Offset

获取缓存区位移。

```
int  Offset() const
```

返回值

缓存区位移

## Offset

设置缓存区位移。

```
void  Offset(
   const int  offset      // 位移
   )
```

参数

*offset*

[输入] 新的缓存区位移。

© 2000-2025, MetaQuotes Ltd.

## 名称

获取缓存区名称。

```
string  Name() const
```

返回值

缓存区名称。

## 名称

设置缓存区名称。

```
void  Name(
   const string  name      // 名称
   )
```

参数

*name*

[输入] 新的缓存区名称。

© 2000-2025, MetaQuotes Ltd.

# At

获取缓存区索引处的元素。

```
double  At(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

指定索引处的缓存区元素。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新整个缓存区。

```
bool  Refresh(
   const int  handle,      // 句柄
   const int  num          // 缓存区号码
   )
```

参数

*handle*

[输入] 指标句柄。

*num*

[输入] 指标缓存区索引。

返回值

true - 如果成功，false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

更新当前(第零号)缓存区中的元素。

```
bool  RefreshCurrent(
   const int  handle,     // 指标句柄
   const int  num         // 缓存区号码
   )
```

参数

*handle*

[输入]指标句柄。

*num*

[输入]缓存区号码。

返回值

true - 如果成功, false - 如果缓存区未能更新。

© 2000-2025, MetaQuotes Ltd.

# CSeries

CSeries 类用于访问标准库里的时间序列数据。

## 描述

CSeries 类可供其所有子类简便地访问 MQL5 时间序列函数。

## 声明

```
class CSeries: public CArrayObj
```

## 标称库文件

```
#include <Indicators\Series.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries

直接分支

CIndicator, CiRealVolume, CiSpread, CiTickVolume, CiTime, CPriceSeries

## 类方法

| 属性 | |
|---|---|
| 名称 | 获取时间序列或指标的名称 |
| BuffersTotal | 获取时间序列或指标缓存区的数量 |
| Timeframe | 获取时间序列或指标的时间帧标志 |
| Symbol | 获取时间序列或指标的品名 |
| Period | 获取时间序列或指标的周期 |
| RefreshCurrent | 获取/设置更新当前数据的标志 |
| 数据访问方法 | |
| virtual BufferResize | 设置时间序列或指标的缓存区大小 |
| 数据更新方法 | |
| virtual Refresh | 更新时间序列或指标的数据 |
| PeriodDescription | 获取周期的字符串描述 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

# 名称

获取时间序列或指标的名称

```
string  Name() const
```

返回值

时间序列或指标的名称。

© 2000-2025, MetaQuotes Ltd.

# BuffersTotal

获取时间序列或指标缓存区的数量。

```
int  BuffersTotal() const
```

返回值

时间序列或指标缓存区的数量。

注释

时间序列仅有一个缓存区。

© 2000-2025, MetaQuotes Ltd.

# Timeframe

获取时间序列或指标的时间帧标志。

```
int  Timeframe() const
```

返回值

时间序列或指标的时间帧标志。

注释

某些时间帧的可见性标志。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获取时间序列或指标的品名。

```
string  Symbol() const
```

返回值

时间序列或指标的品名。

© 2000-2025, MetaQuotes Ltd.

# Period

获取时间序列或指标的周期。

```
ENUM_TIMEFRAMES  Period() const
```

返回值

时间序列或指标的周期（值为 ENUM_TIMEFRAMES 枚举）。

© 2000-2025, MetaQuotes Ltd.

# RefreshCurrent

设置更新时间序列或指标当前值的标志。

```
string  RefreshCurrent(
   const bool  flag      // 新标志
   )
```

参数

*flag*

[输入] 新标志。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# BufferSize

返回时间序列或指标缓存区内可用数据的数量。

```
int  BufferSize() const
```

返回值

时间序列或指标缓存区内可用数据的数量。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置时间序列或指标的缓存区大小。

```
virtual bool  BufferResize(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 缓存区的新大小。

返回值

true — 如果成功，否则 false。

注释

所有时间序列和指标缓存区有相同的大小。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列或指标的数据。

```
virtual void  Refresh(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 时间帧更新 (标志)。

© 2000-2025, MetaQuotes Ltd.

# PeriodDescription

获取指定 ENUM_TIMEFRAMES 枚举值的字符串表述。

```
string  PeriodDescription(
   const int  val=0      // 值
   )
```

参数

*val=0*

[输入] 转换值。

返回值

指定 ENUM_TIMEFRAMES 枚举值的字符串表述。

注释

如果未指定值或值为零，它返回时间序列或指标的时间帧。

© 2000-2025, MetaQuotes Ltd.

# CPriceSeries

CPriceSeries 是访问价格数据的基准类。

## 描述

CSeries 类可供其所有子类简便地访问 MQL5 函数来操作价格数据。

## 声明

```
class CPriceSeries: public CSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CPriceSeries

直接分支

CiClose, CiHigh, CiLow, CiOpen

## 类方法

| 创建方法 | |
|---|---|
| virtual BufferResize | 设置缓存区大小 |
| 数据访问方法 | |
| virtual GetData | 获取缓存区指定索引处的元素 |
| 数据更新方法 | |
| virtual Refresh | 更新时间序列数据 |
| 数据搜索方法 | |
| virtual MinIndex | 获取指定范围最小元素索引 |
| virtual MinValue | 获取指定范围最小元素值和索引 |
| virtual MaxIndex | 获取指定范围最大元素索引 |
| virtual MaxValue | 获取指定范围最大元素值和索引 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置缓存区新大小。

```
virtual void  BufferResize(
   const int  size      // 新大小
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

# GetData

获取缓存区指定索引处的元素。

```
double  GetData(
   const int  index      // 索引
   ) const
```

参数

*index*

[输入] 缓存区元素的索引。

返回值

缓存区指定索引处的元素，或者 EMPTY_VALUE。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列的数据

```
virtual void  Refresh(
   const int  flags=OBJ_ALL_PERIODS      // 时间帧标志
   )
```

参数

*flags=OBJ_ALL_PERIODS*

[输入] 时间帧更新 (标志)。

© 2000-2025, MetaQuotes Ltd.

# MinIndex

获取指定范围最小元素索引。

```
virtual int  MinIndex(
   const int  start,      // 开始索引
   const int  count       // 扫描的元素数量
   ) const
```

参数

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

返回值

指定范围最小元素索引，或者在出错时 -1。

© 2000-2025, MetaQuotes Ltd.

# MinValue

获取指定范围最小元素值和索引。

```
virtual double  MinValue(
   const int    start,      // 开始索引
   const int    count,      // 扫描的元素数量
   int&         index       // 索引变量的引用
   ) const
```

参数

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

*index*

[输出] 整数类型变量的引用。

返回值

指定范围指定缓存区的最小元素值，或者错误时 EMPTY_VALUE。

注释

最小元素的索引保存在索引变量中。

© 2000-2025, MetaQuotes Ltd.

# MaxIndex

获取指定范围最大元素索引。

```
virtual int  MaxIndex(
   const int  start,      // 开始索引
   const int  count       // 扫描的元素数量
   ) const
```

参数

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

返回值

指定范围最大元素索引，或者在出错时 -1。

© 2000-2025, MetaQuotes Ltd.

# MaxValue

获取指定范围最大元素值和索引。

```
virtual double  MaxValue(
   const int    start,      // 开始索引
   const int    count,      // 扫描的元素数量
   int&         index       // 索引变量的引用
   ) const
```

参数

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

*index*

[输出] 整数类型变量的引用。

返回值

指定范围指定缓存区的最大元素值，或者错误时 EMPTY_VALUE。

注释

最大元素的索引保存在索引变量中。

© 2000-2025, MetaQuotes Ltd.

# CIndicator

CIndicator是MQL5标准库中的所有技术指标类的基准类。

## 描述

CIndicator类可供其所有子类简便地访问MQL5 API技术指标函数。

## 声明

```
class CIndicator: public CSeries
```

## 标称库文件

```
#include <Indicators\Indicator.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator

直接分支

CiAC, CiAD, CiADX, CiADXWilder, CiAlligator, CiAMA, CiAO, CiATR, CiBands, CiBearsPower, CiBullsPower, CiBWMFI, CiCCI, CiChaikin, CiCustom, CiDEMA, CiDeMarker, CiEnvelopes, CiForce, CiFractals, CiFrAMA, CiGator, CiIchimoku, CiMA, CiMACD, CiMFI, CiMomentum, CiOBV, CiOsMA, CiRSI, CiRVI, CiSAR, CiStdDev, CiStochastic, CiTEMA, CiTriX, CiVIDyA, CiVolumes, CiWPR

## 类方法

| 属性 | |
|---|---|
| Handle | 获取指标的句柄。 |
| Status | 获取指标的状态。 |
| FullRelease | 设置释放指标句柄的标志。 |
| 创建 | |
| 创建 | 创建指标 |
| BufferResize | 设置新的缓存区大小 |
| 数据访问方法 | |
| GetData | 从指标的缓存区拷贝数据 |
| 数据更新方法 | |
| Refresh | 更新指标的数据 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| 查询最小/最大值 | |
| Minimum | 获取指定范围指定缓存区的最小元素索引。 |
| MinValue | 获取指定范围指定缓存区的最小元素的值和索引。 |
| Maximum | 获取指定范围指定缓存区的最大元素索引。 |
| MaxValue | 获取指定范围指定缓存区的最大元素值和索引。 |
| 枚举转换 | |
| MethodDescription | 获取 ENUM_MA_METHOD 枚举值的字符串描述 |
| PriceDescription | 获取 ENUM_APPLIED_PRICE 枚举值的字符串描述 |
| VolumeDescription | 获取 ENUM_APPLIED_VOLUME 枚举值的字符串描述 |
| 操作图表 | |
| AddToChart | 在图表上加入指标 |
| DeleteFromChart | 从图表上删除指标 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

派生类:

- CiAC
- CiAD
- CiADX
- CiADXWilder
- CiAlligator
- CiAMA
- CiAO
- CiATR
- CiBands

© 2000-2025, MetaQuotes Ltd.

- CiBearsPower
- CiBullsPower
- CiBWMFI
- CiCCI
- CiChaikin
- CiDEMA
- CiDeMarker
- CiEnvelopes
- CiForce
- CiFractals
- CiFrAMA
- CiGator
- CiIchimoku
- CiMA
- CiMACD
- CiMFI
- CiMomentum
- CiOBV
- CiOsMA
- CiRSI
- CiRVI
- CiSAR
- CiStdDev
- CiStochastic
- CiTEMA
- CiTriX
- CiVIDyA
- CiVolumes
- CiWPR

© 2000-2025, MetaQuotes Ltd.

# Handle

获取指标的句柄。

```
int  Handle() const
```

返回值

指标句柄。

© 2000-2025, MetaQuotes Ltd.

# Status

获取指标的状态。

```
string  Status() const
```

返回值

指标创建标志。

© 2000-2025, MetaQuotes Ltd.

# FullRelease

设置释放指标句柄的标志。

```
void  FullRelease(
   const bool  flag=true      // 标志
   )
```

参数

*flag*

[输入] 句柄释放的标志新值。

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建指标。

```
bool  Create(
   const string           symbol,          // 品名
   const ENUM_TIMEFRAMES  period,          // 周期
   const ENUM_INDICATOR   type,            // 类型
   const int              num_params,      // 参数数量
   const MqlParam&        params[]         // 参数数组的引用
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 周期 (ENUM_TIMEFRAMES 枚举)。

*type*

[输入] 指标类型 (ENUM_INDICATOR 枚举)。

*num_params*

[输入] 指标参数的数量。

*params*

[输入] 用于指标的参数数组的引用。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置指标的缓存区大小。

```
virtual bool  BufferResize(
   const int  size      // 大小
   )
```

参数

*size*

[输入] 新缓存区大小。

返回值

true — 如果成功，否则 false。

注释

所有指标缓存区有相同的大小。

© 2000-2025, MetaQuotes Ltd.

# BarsCalculated

返回指标已计算数据的数量。

```
int  BarsCalculated() const;
```

返回值

返回在指标缓存区里已计算数据的额度，或者在出错时 -1 (数据尚未计算)。

© 2000-2025, MetaQuotes Ltd.

## GetData

从指标的指定缓存区获取指定元素。Refresh() 方法应在使用该方法之前调用，以便得到最近的数据。

```
double  GetData(
   const int  buffer_num,     // 缓存区号码
   const int  index           // 元素索引
   ) const
```

参数

*buffer_num*

[输入] 缓存区号码。

*index*

[输入] 元素索引。

返回值

如果成功，它返回元素的数值，或在出错情况下 EMPTY_VALUE。

## GetData

从指标的缓存区获取从开始位置到必要数据长度的数据。

```
int  GetData(
   const int      start_pos,      // 位置
   const int      count,          // 所需元素长度
   const int      buffer_num,     // 缓存区号码
   double&        buffer[]        // 数据的目标数组
   ) const
```

参数

*start_pos*

[输入] 指标缓存区的开始位置。

*count*

[输入] 所需元素的数量。

*buffer_num*

[输入] 指标缓存区的号码。

*buffer*

[输入] 目标数据数组的引用。

返回值

如果成功，返回从指定指标缓存区接收元素的数量，否则 -1。

## GetData

从指标的缓存区获取从开始时间到必要数据长度的数据。

© 2000-2025, MetaQuotes Ltd.

```
int  GetData(
   const datetime  start_time,      // 开始时间
   const int       count,           // 所需元素长度
   const int       buffer_num,      // 缓存区号码
   double&         buffer[]         // 数据的目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer_num*

[输入] 指标缓存区的号码。

*buffer*

[输入] 目标数组的引用。

返回值

如果成功，返回从指定指标缓存区接收元素的数量，否则 -1。

## GetData

从指标的缓存区获取从开始时间到截止时间以及必要数据长度的数据。

```
int  GetData(
   const datetime  start_time,      // 开始时间
   const datetime  stop_time,       // 截止时间
   const int       buffer_num,      // 缓存区号码
   double&         buffer[]         // 数据的目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer_num*

[输入] 指标缓存区的号码。

*buffer*

[输入] 目标数组的引用。

返回值

如果成功，返回从指定指标缓存区接收元素的数量，否则 -1。

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新指标的数据。建议在使用 GetData() 方法前调用。

```
virtual void  Refresh(
   int  flags=OBJ_ALL_PERIODS      // 标志
   )
```

参数

*flags=OBJ_ALL_PERIODS*

[输入] 时间帧更新标志。

© 2000-2025, MetaQuotes Ltd.

# Minimum

返回指定范围指定缓存区的最小元素索引。

```
int  Minimum(
   const int  buffer_num,      // 缓存区号码
   const int  start,           // 开始索引
   const int  count            // 处理元素的数量
   ) const
```

参数

*buffer_num*

[输入] 搜索数值所在的缓存区号码。

*start*

[输入] 搜索开始的索引。

*count*

[输入] 搜索元素的数量。

返回值

指定范围指定缓存区的最小元素索引。

© 2000-2025, MetaQuotes Ltd.

# MinValue

返回指定范围指定缓存区的最小元素值和索引。

```
double  MinValue(
   const int    buffer_num,      // 缓存区号码
   const int    start,           // 开始索引
   const int    count,           // 处理元素的数量
   int&         index            // 引用
   ) const
```

参数

*buffer_num*

[输入] 搜索数值所在的缓存区号码。

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

*index*

[输出] 用于最小元素索引的整数型变量的引用。

返回值

指定范围指定缓存区的最小元素值。

注释

最小缓存区元素的索引保存在索引变量中，通过引用传递。

© 2000-2025, MetaQuotes Ltd.

# Maximum

返回指定范围指定缓存区的最大元素索引。

```
int  Maximum(
   const int  buffer_num,      // 缓存区号码
   const int  start,           // 开始索引
   const int  count            // 处理元素的数量
   ) const
```

参数

*buffer_num*

[输入] 搜索数值所在的缓存区号码。

*start*

[输入] 搜索开始的索引。

*count*

[输入] 搜索元素的数量。

返回值

指定范围指定缓存区的最大元素索引。

© 2000-2025, MetaQuotes Ltd.

# MaxValue

返回指定范围指定缓存区的最大元素值和索引。

```
double  MaxValue(
   const int    buffer_num,      // 缓存区号码
   const int    start,           // 开始索引
   const int    count,           // 处理元素的数量
   int&         index            // 引用
   ) const
```

参数

*buffer_num*

[输入] 搜索数值所在的缓存区号码。

*start*

[输入] 开始索引。

*count*

[输入] 处理元素的数量。

*index*

[输出] 用于最大元素索引的整数型变量的引用。

返回值

指定范围指定缓存区的最大元素值。

注释

最大缓存区元素的索引保存在索引变量中，通过引用传递。

© 2000-2025, MetaQuotes Ltd.

# MethodDescription

函数返回 ENUM_MA_METHOD 枚举值的字符串描述。

```
string  MethodDescription(
   const int  val      // 值
   ) const
```

参数

*val*

[输入] ENUM_MA_METHOD 枚举值。

返回值

返回 ENUM_MA_METHOD 枚举值的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# PriceDescription

此方法返回 ENUM_APPLIED_PRICE 枚举值的字符串描述。

```
string  PriceDescription(
   const int  val      // 值
   ) const
```

参数

*val*

[输入] 值为 ENUM_APPLIED_PRICE 枚举。

返回值

返回 ENUM_APPLIED_PRICE 枚举值的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# VolumeDescription

此方法返回 ENUM_APPLIED_VOLUME 枚举值的字符串描述。

```
string  VolumeDescription(
   const int  val      // 值
   ) const
```

参数

*val*

[输入] ENUM_APPLIED_VOLUME 枚举值。

返回值

返回 ENUM_APPLIED_VOLUME 枚举值的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# AddToChart

在图表上加入指标。

```
bool  AddToChart(
   const long  chart,      // 图表 ID
   const int  subwin       // 图表子窗口
   )
```

参数

*chart*

[输入] 图表 ID。

*subwin*

[输入] 图表子窗口。

返回值

true — 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# DeleteFromChart

从图表上删除指标。

```
bool  DeleteFromChart(
   const long  chart,      // 图表 ID
   const int  subwin       // 图表子窗口
   )
```

参数

*chart*

[输入] 图表 ID。

*subwin*

[输入] 图表子窗口。

返回值

true — 如果成功，false 如果出错。

© 2000-2025, MetaQuotes Ltd.

# CIndicators

CIndicators 类用于收集时间序列和技术指标类实例。

## 描述

CIndicators 类可供创建技术指标类实例，它们的存储和管理（数据同步，处理以及资金管理）。

## 声明

```
class CIndicators: public CArrayObj
```

## 标称库文件

```
#include <Indicators\Indicators.mqh>
```

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建技术指标 |
| 数据更新方法 | |
| Refresh | 更新所有集合内的技术指标数据。 |

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。

```
CIndicator*  Create(
   const string            symbol,      // 品名
   const ENUM_TIMEFRAMES   period,      // 周期
   const ENUM_INDICATOR    type,        // 指标类型
   const int               count,       // 参数数量
   const MqlParam&         params       // 参数数组的引用
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 周期 (ENUM_TIMEFRAMES 枚举)。

*type*

[输入] 指标类型 (ENUM_INDICATOR)。

*count*

[输入] 指标的参数数量。

*params*

[输入] 用于指标的参数数组的引用。

返回值

如果成功，它返回所创建指标的引用，如果指标未能创建，则 NULL。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新所有集合内的技术指标。

```
int  Refresh()
```

返回值

它返回更新的时间帧标志 (形成一个对象可见性标志)。

© 2000-2025, MetaQuotes Ltd.

# 时间序列类

这一组章节包含 MQL5 标准库内时间序列类的技术细节，以及它的所有关键组件的描述。

| 类 | 描述 |
| --- | --- |
| CiSpread | 可供访问点差历史数据 |
| CiTime | 可供访问历史柱线的开盘时间 |
| CiTickVolume | 可供访问历史柱线的报价交易量 |
| CiRealVolume | 可供访问历史柱线的实际交易量 |
| CiOpen | 可供访问历史柱线的开盘价 |
| CiHigh | 可供访问历史柱线的最高价 |
| CiLow | 可供访问历史柱线的最低价 |
| CiClose | 可供访问历史柱线的收盘价 |

© 2000-2025, MetaQuotes Ltd.

# CiSpread

CiSpread 类设计用于访问历史柱线的点差。

## 描述

CiSpread 类可供访问点差历史数据

## 声明

```
class CiSpread: public CSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CiSpread

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| BufferResize | 设置缓存区大小 |
| 数据访问方法 | |
| GetData | 获取数据 |
| 数据更新方法 | |
| Refresh | 更新数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问点差历史。

```
bool  Create(
   string           symbol,      // 品名
   ENUM_TIMEFRAMES  period       // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置序列新大小。

```
virtual void  BufferResize(
   int  size      // 新值
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内指定索引处的元素。

```
int  GetData(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 所需元素的索引。

返回值

时间序列缓存区元素，或 0。

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int   start_pos,      // 起始位置
   int   count,          // 获取元素的数量
   int&  buffer          // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   int&      buffer           // 目标数组
   ) const
```

参数

*start_time*

© 2000-2025, MetaQuotes Ltd.

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   int&      buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列的数据。

```
virtual void  Refresh(
   int  flags      // 标志
   )
```

参数

*flags*

[输入] 时间帧标志。

© 2000-2025, MetaQuotes Ltd.

# CiTime

CiTime 类设计用于访问历史柱线的开盘时间。

## 描述

CiTime 类可供访问历史柱线的开盘时间。

## 声明

```
class CiTime: public CSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CiTime

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| BufferResize | 设置缓存区大小 |
| 数据访问方法 | |
| GetData | 获取数据 |
| 数据更新方法 | |
| Refresh | 更新数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的开盘时间。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功，false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置序列新大小。

```
virtual void  BufferResize(
   int  size      // 新值
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内指定索引处的元素。

```
datetime  GetData(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 所需元素的索引。

返回值

时间序列缓存区元素，或 0。

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int   start_pos,      // 起始位置
   int   count,          // 获取元素的数量
   long& buffer          // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   long&     buffer           // 目标数组
   ) const
```

参数

*start_time*

© 2000-2025, MetaQuotes Ltd.

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   long&     buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列的数据。

```
virtual void  Refresh(
   int  flags      // 标志
   )
```

参数

*flags*

[输入] 时间帧标志。

© 2000-2025, MetaQuotes Ltd.

# CiTickVolume

CiTickVolume 类设计用于访问历史柱线的报价交易量。

## 描述

CiTickVolume 类可供访问历史柱线的报价交易量。

## 声明

```
class CiTickVolume: public CSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CiTickVolume

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| BufferResize | 设置缓存区大小 |
| 数据访问方法 | |
| GetData | 获取数据 |
| 数据更新方法 | |
| Refresh | 更新数据 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的报价交易量。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置序列新大小。

```
virtual void  BufferResize(
   int  size      // 新值
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内指定索引处的元素。

```
long  GetData(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 所需元素的索引。

返回值

时间序列缓存区元素，或 0。

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int   start_pos,     // 起始位置
   int   count,         // 获取元素的数量
   long& buffer         // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,     // 开始时间
   int       count,          // 元素的数量
   long&     buffer          // 目标数组
   ) const
```

参数

*start_time*

© 2000-2025, MetaQuotes Ltd.

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

```
int  GetData(
   datetime  start_time,     // 开始时间
   datetime  stop_time,      // 截止时间
   long&     buffer          // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列的数据。

```
virtual void  Refresh(
   int  flags      // 标志
   )
```

参数

*flags*

[输入] 时间帧标志。

© 2000-2025, MetaQuotes Ltd.

# CiRealVolume

CiRealVolume 类设计用于访问历史柱线的实际交易量。

## 描述

CiRealVolume 类可供访问历史柱线的实际交易量。

## 声明

```
class CiRealVolume: public CSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CiRealVolume

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| BufferResize | 设置缓存区大小 |
| 数据访问方法 | |
| GetData | 获取数据 |
| 数据更新方法 | |
| Refresh | 更新数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的实际交易量。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

# BufferResize

设置序列新大小。

```
virtual void  BufferResize(
   int  size      // 新值
   )
```

参数

*size*

[输入] 新缓存区大小。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内指定索引处的元素。

```
datetime  GetData(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 所需元素的索引。

返回值

时间序列缓存区元素，或 0。

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int   start_pos,      // 起始位置
   int   count,          // 获取元素的数量
   long& buffer          // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   long&     buffer           // 目标数组
   ) const
```

参数

*start_time*

© 2000-2025, MetaQuotes Ltd.

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   long&     buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新时间序列的数据。

```
virtual void  Refresh(
   int  flags      // 标志
   )
```

参数

*flags*

[输入] 时间帧标志。

© 2000-2025, MetaQuotes Ltd.

# CiOpen

CiOpen 类设计用于访问历史柱线的开盘价。

## 描述

CiOpen 类可供访问历史柱线的开盘价。

## 声明

```
class CiOpen: public CPriceSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CPriceSeries
          CiOpen

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| 数据访问方法 | |
| GetData | 获取数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类 CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的开盘价。

```
bool  Create(
   string           symbol,      // 品名
   ENUM_TIMEFRAMES  period       // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int    start_pos,      // 起始位置
   int    count,          // 获取元素的数量
   double& buffer         // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   double&     buffer         // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

© 2000-2025, MetaQuotes Ltd.

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   double&   buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# CiHigh

CiHigh 类设计用于访问历史柱线的最高价。

## 描述

CiHigh 类可供访问历史柱线的最高价。

## 声明

```
class CiHigh: public CPriceSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CPriceSeries
CiHigh

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| 数据访问方法 | |
| GetData | 获取数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类 CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的最高价。

```
bool  Create(
   string           symbol,      // 品名
   ENUM_TIMEFRAMES  period       // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int    start_pos,      // 起始位置
   int    count,          // 获取元素的数量
   double& buffer         // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   double&     buffer         // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

© 2000-2025, MetaQuotes Ltd.

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   double&   buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# CiLow

CiLow 类设计用于访问历史柱线的最低价。

## 描述

CiLow 类可供访问历史柱线的最低价。

## 声明

```
class CiLow: public CPriceSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CPriceSeries
CiLow

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| 数据访问方法 | |
| GetData | 获取数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类 CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的最低价。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int    start_pos,      // 起始位置
   int    count,          // 获取元素的数量
   double& buffer         // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   double&     buffer         // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

© 2000-2025, MetaQuotes Ltd.

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   double&   buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# CiClose

CiClose 类设计用于访问历史柱线的收盘价。

## 描述

CiClose 类可供访问历史柱线的收盘价。

## 声明

```
class CiClose: public CPriceSeries
```

## 标称库文件

```
#include <Indicators\TimeSeries.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CPriceSeries
          CiClose

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建时间序列 |
| 数据访问方法 | |
| GetData | 获取数据 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类 CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

© 2000-2025, MetaQuotes Ltd.

# 创建

以指定参数创建时间序列以便访问历史柱线的收盘价。

```
bool  Create(
   string           symbol,      // 品名
   ENUM_TIMEFRAMES  period       // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果时间序列未能创建。

© 2000-2025, MetaQuotes Ltd.

## GetData

获取时间序列内从开始位置到必要元素长度的数据。

```
int  GetData(
   int    start_pos,      // 起始位置
   int    count,          // 获取元素的数量
   double& buffer         // 目标数组
   ) const
```

参数

*start_pos*

[输入] 时间序列的开始位置。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到必要元素长度的数据。

```
int  GetData(
   datetime  start_time,      // 开始时间
   int       count,           // 元素的数量
   double&     buffer         // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*count*

[输入] 所需元素的数量。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

## GetData

获取时间序列内从开始时间到截止时间的数据。

© 2000-2025, MetaQuotes Ltd.

```
int  GetData(
   datetime  start_time,      // 开始时间
   datetime  stop_time,       // 截止时间
   double&   buffer           // 目标数组
   ) const
```

参数

*start_time*

[输入] 开始时间。

*stop_time*

[输入] 截止时间。

*buffer*

[输入] 目标数据数组的引用。

返回值

>=0 如果成功，-1 出错情况下。

© 2000-2025, MetaQuotes Ltd.

# 趋势指标类

这一组章节包含 MQL5 标准库中趋势指标类的技术细节，以及它的所有关键组件的描述。

| 类/组 | 描述 |
| --- | --- |
| CiADX | 平均放线指标 |
| CiADXWilder | Welles Wilder平均方向指数 |
| CiBands | 布林带® |
| CiEnvelopes | 包络线 |
| CiIchimoku | Ichimoku Kinko Hyo |
| CiMA | 移动均线 |
| CiSAR | 抛物线停止和转向系统 |
| CiStdDev | 标准方差 |
| CiDEMA | 双重指数移动均线 |
| CiTEMA | 三重指数移动均线 |
| CiFrAMA | 分形自适应移动均线 |
| CiAMA | 自适应移动均线 |
| CiVIDyA | 可变指数动态均线 |

© 2000-2025, MetaQuotes Ltd.

# CiADX

CiADX 类旨在使用平均方向指数技术指标。

## 描述

CiADX 类可供创建，设置和访问平均方向指标的数据。

## 声明

```
class CiADX: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiADX

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回主线缓存区元素 |
| Plus | 返回 +DI线缓存区元素 |
| Minus | 返回 -DI线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Plus

返回 -DI线指定索引的缓存元素。

```
double   Plus(
   int   index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 +DI 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Minus

返回 +DI线指定索引的缓存元素。

```
double  Minus(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 +DI 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_ADX 对应 CiADX)。

© 2000-2025, MetaQuotes Ltd.

# CiADXWilder

CiADXWilder类旨在使用WellesWilder平均方向指数技术指标。

## 描述

CiADXWilder类可供创建，设置和访问WellesWilder平均方向指标的数据。

## 声明

```
class CiADXWilder: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiADXWilder

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回主线缓存区元素 |
| Plus | 返回+DI线缓存区元素 |
| Minus | 返回-DI线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Plus

返回 -DI线指定索引的缓存元素。

```
double   Plus(
   int   index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 +DI 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Minus

返回 +DI线指定索引的缓存元素。

```
double  Minus(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 +DI 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_ADXW 对应 CiADXWilder)。

© 2000-2025, MetaQuotes Ltd.

# CiBands

CiBands 类旨在使用布林带® 技术指标。

## 描述

CiBands 类可供创建，设置和访问布林带指标的数据。

## 声明

```
class CiBands: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiBands

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| MaShift | 返回水平平移 |
| Deviation | 返回方差 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Base | 返回基准线缓存区元素 |
| Upper | 返回上边界线缓存区元素 |
| Lower | 返回下边界线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaShift

返回指标的水平平移。

```
int  MaShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Deviation

返回方差。

```
double  Deviation() const
```

返回值

返回指标创建时定义的方差。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,          // 品名
   ENUM_TIMEFRAMES   period,          // 周期
   int               ma_period,       // 平均周期
   int               ma_shift,        // 平移
   double            deviation,       // 方差
   int               applied          // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_shift*

[输入] 指标的水平平移。

*deviation*

[输入] 方差。

*applied*

[输入] 交易量应用类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Base

返回基准线指定索引的缓存元素。

```
double  Base(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

基准线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Upper

返回上边界线指定索引的缓存元素。

```
double  Upper(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

上边界线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Lower

返回下边界线指定索引的缓存元素。

```
double  Lower(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

下边界线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_BANDS 对应 CiBands)。

© 2000-2025, MetaQuotes Ltd.

# CiEnvelopes

CiEnvelopes 类旨在使用包络线技术指标。

## 描述

CiEnvelopes 类可供创建，设置和访问包络线指标的数据。

## 声明

```
class CiEnvelopes: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiEnvelopes

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| MaShift | 返回水平平移 |
| MaMethod | 返回平均方法 |
| Deviation | 返回方差 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Upper | 返回上边界线缓存区元素 |
| Lower | 返回下边界线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaShift

返回指标的水平平移。

```
int  MaShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# Deviation

返回偏差的值。

```
double   Deviation() const
```

返回值

返回指标创建时定义的偏差值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,        // 品名
   ENUM_TIMEFRAMES   period,        // 周期
   int               ma_period,     // 平均周期
   int               ma_shift,      // 水平平移
   ENUM_MA_METHOD    ma_method,     // 平均方法
   int               applied,       // 处理或应用的价格类型
   double            deviation      // 偏差
   )
```

参数

*symbol*
[输入] 品名。

*period*
[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*
[输入] 平均周期。

*ma_shift*
[输入] 水平平移。

*ma_method*
[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*
[输入] 处理或应用的价格类型。

*deviation*
[输入] 方差。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Upper

返回上边界线指定索引的缓存元素。

```
double  Upper(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

上边界线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Lower

返回下边界线指定索引的缓存元素。

```
double  Lower(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

下边界线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_ENVELOPES 对应 CiEnvelopes)。

© 2000-2025, MetaQuotes Ltd.

# CiIchimoku

CiIchimoku 类旨在使用 Ichimoku Kinko Hyo 技术指标。

## 描述

CiIchimoku 类可供创建，设置和访问 Ichimoku Kinko Hyo 指标的数据。

## 声明

```
class CiIchimoku: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiIchimoku

## 类方法

| 属性 | |
|---|---|
| TenkanSenPeriod | 返回 TenkanSen 周期 |
| KijunSenPeriod | 返回 KijunSen 周期 |
| SenkouSpanBPeriod | 返回 SenkouSpanB 周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| TenkanSen | 返回 TenkanSen 线缓存区元素 |
| KijunSen | 返回 KijunSen 线缓存区元素 |
| SenkouSpanA | 返回 SenkouSpanA 线缓存区元素 |
| SenkouSpanB | 返回 SenkouSpanB 线缓存区元素 |
| ChinkouSpan | 返回 ChikouSpan 线缓存区元素 |
| 输入/输出 | |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# TenkanSenPeriod

返回 TenkanSen 周期。

```
int  TenkanSenPeriod() const
```

返回值

返回指标创建时定义的 TenkanSen 周期。

© 2000-2025, MetaQuotes Ltd.

# KijunSenPeriod

返回 KijunSen 周期。

```
int  KijunSenPeriod() const
```

返回值

返回指标创建时定义的 KijunSen 周期。

© 2000-2025, MetaQuotes Ltd.

# SenkouSpanBPeriod

返回 SenkouSpanB 周期。

```
int  SenkouSpanBPeriod() const
```

返回值

返回指标创建时定义的 SenkouSpanB 周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,           // 品名
   ENUM_TIMEFRAMES  period,           // 周期
   int              tenkan_sen,       // 周期 TenkanSen
   int              kijun_sen,        // 周期 KijunSen
   int              senkou_span_b     // 周期 SenkouSpanB
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*tenkan_sen*

[输入] TenkanSen 周期。

*kijun_sen*

[输入] KijunSen 周期。

*senkou_span_b*

[输入] SenkouSpanB 周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# TenkanSen

返回 TenkanSen 线指定索引的缓存元素。

```
double  TenkanSen(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

TenkanSen 线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# KijunSen

返回 KijunSen 线指定索引的缓存元素。

```
double  KijunSen(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 KijunSen 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# SenkouSpanA

返回 SenkouSpanA 线指定索引的缓存元素。

```
double  SenkouSpanA(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 SenkouSpanA 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# SenkouSpanB

返回 SenkouSpanB 线指定索引的缓存元素。

```
double  SenkouSpanB(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

指定索引处的 SenkouSpanB 线缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# ChinkouSpan

返回 ChinkouSpan 线指定索引的缓存元素。

```
double  ChinkouSpan(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

ChinkouSpan 线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_ICHIMOKU 对应 CiIchimoku）。

© 2000-2025, MetaQuotes Ltd.

# CiMA

CiMA 类旨在使用移动均线技术指标。

## 描述

CiMA 类可供创建，设置和访问移动均线指标的数据。

## 声明

```
class CiMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiMA

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| MaShift | 返回水平平移 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaShift

返回指标的水平平移。

```
int  MaShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回指标创建时定义的平均方法（值为 ENUM_MA_METHOD 枚举）。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            string,        // 品名
   ENUM_TIMEFRAMES   period,        // 周期
   int               ma_period,     // 平均周期
   int               ma_shift,      // 水平平移
   ENUM_MA_METHOD    ma_method,     // 平均方法
   int               applied        // 处理或应用的价格类型
   )
```

参数

*string*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_shift*

[输入] 水平平移。

*ma_method*

[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_MA 对应 CiMA）。

© 2000-2025, MetaQuotes Ltd.

# CiSAR

CiSAR 类旨在使用抛物线停止和转向技术指标。

## 描述

CiSAR 类可供创建，设置和访问抛物线停止和转向指标的数据。

## 声明

```
class CiSAR: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiSAR

## 类方法

| 属性 | |
|---|---|
| SarStep | 返回速度增长的步长 |
| Maximum | 返回价格跟随的系数 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# SarStep

返回速度增长的步长 (加速系数)。

```
double  SarStep() const
```

返回值

回指标创建时定义的速度增长的步长。

© 2000-2025, MetaQuotes Ltd.

# Maximum

返回价格跟随的系数。

```
double  Maximum() const
```

返回值

在指标创建时定义的价格跟随的系数。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,      // 品名
   ENUM_TIMEFRAMES  period,      // 周期
   double           step,        // 步长
   double           maximum      // 系数
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*step*

[输入] 速度增长的步长

*maximum*

[输入] 价格跟随系数。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_SAR 对应 CiSAR）。

© 2000-2025, MetaQuotes Ltd.

# CiStdDev

CiStdDev 类旨在使用标准方差技术指标。

## 描述

CiStdDev 类可供创建，设置和访问标准方差指标的数据。

## 声明

```
class CiStdDev: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiStdDev

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| MaShift | 返回水平平移 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaShift

返回指标的水平平移。

```
int  MaShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回指标创建时定义的平均方法（值为 ENUM_MA_METHOD 枚举）。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,          // 品名
   ENUM_TIMEFRAMES   period,          // 周期
   int               ma_period,       // 平均周期
   int               ma_shift,        // 水平平移
   ENUM_MA_METHOD    ma_method,       // 平均方法
   int               applied          // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_shift*

[输入] 水平平移。

*ma_method*

[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_STDDEV 对应 CiStdDev)。

© 2000-2025, MetaQuotes Ltd.

# CiDEMA

CiDEMA 类旨在使用双重指数移动均线技术指标。

## 描述

CiDEMA 类可供创建，设置和访问双重指数移动均线指标的数据。

## 声明

```
class CiDEMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiDEMA

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| IndShift | 返回水平平移 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# IndShift

返回指标的水平平移。

```
int  IndShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           string,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period,     // 平均周期
   int              ind_shift,     // 平移
   int              applied        // 处理或应用的价格类型
   )
```

参数

*string*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ind_shift*

[输入] 水平平移。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_DEMA 对应 CiDEMA)。

© 2000-2025, MetaQuotes Ltd.

# CiTEMA

CiTEMA 类旨在使用三重指数移动均线技术指标。

## 描述

CiTEMA 类可供创建，设置和访问三重指数移动均线指标的数据。

## 声明

```
class CiTEMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiTEMA

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| IndShift | 返回水平平移 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# IndShift

返回指标的水平平移。

```
int  IndShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,          // 品名
   ENUM_TIMEFRAMES   period,          // 周期
   int               ma_period,       // 平均周期
   int               ma_shift,        // 位移
   int               applied          // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_shift*

[输入] 水平平移。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_TEMA 对应 CiTEMA)。

© 2000-2025, MetaQuotes Ltd.

# CiFrAMA

CiFrAMA 类旨在使用分形自适应移动均线技术指标。

## 描述

CiFrAMA 类可供创建，设置和访问分形自适应移动均线指标的数据。

## 声明

```
class CiFrAMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiFrAMA

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| IndShift | 返回水平平移 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# IndShift

返回指标的水平平移。

```
int  IndShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,          // 品名
   ENUM_TIMEFRAMES  period,          // 周期
   int              ma_period,       // 平均周期
   int              ma_shift,        // 位移
   int              applied          // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_shift*

[输入] 水平平移。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_FRAMA 对应 CiFrAMA）。

© 2000-2025, MetaQuotes Ltd.

# CiAMA

CiAMA 类旨在使用自适应移动均线技术指标。

## 描述

CiAMA 类可供创建，设置和访问自适应移动均线指标的数据。

## 声明

```
class CiAMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiAMA

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| FastEmaPeriod | 返回快速 EMA 线的平均周期。 |
| SlowEmaPeriod | 返回慢速 EMA 线的平均周期。 |
| IndShift | 返回水平平移 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# FastEmaPeriod

返回快速 EMA 线的平均周期。

```
int  FastEmaPeriod() const
```

返回值

返回指标创建时定义的快速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SlowEmaPeriod

返回慢速 EMA 线的平均周期。

```
int  SlowEmaPeriod() const
```

返回值

返回指标创建时定义的慢速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# IndShift

返回指标的水平平移。

```
int  IndShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           string,              // 品名
   ENUM_TIMEFRAMES  period,              // 周期
   int              ma_period,           // 平均周期
   int              fast_ema_period,     // 快速 EMA 周期
   int              slow_ema_period,     // 慢速 EMA 周期
   int              ind_shift,           // 平移
   int              applied              // 处理或应用的价格类型
   )
```

参数

*string*
[输入] 品名。

*period*
[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*
[输入] 平均周期。

*fast_ema_period*
[输入] 快速 EMA 平均周期。

*slow_ema_period*
[输入] 慢速 EMA 平均周期。

*ind_shift*
[输入] 水平平移。

*applied*
[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_AMA 对应 CiAMA)。

© 2000-2025, MetaQuotes Ltd.

# CiVIDyA

CiVIDyA 类旨在使用可变指数动态均线技术指标。

## 描述

CiVIDyA 类可供创建，设置和访问可变指数动态均线指标的数据。

## 声明

```
class CiVIDyA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Trend.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiVIDyA

## 类方法

| 属性 | |
|---|---|
| CmoPeriod | 返回动量周期。 |
| EmaPeriod | 返回 EMA 线的平均周期。 |
| IndShift | 返回水平平移 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# CmoPeriod

返回动量周期。

```
int  CmoPeriod() const
```

返回值

返回指标创建时定义的计算动量周期。

© 2000-2025, MetaQuotes Ltd.

# EmaPeriod

返回 EMA 线的平均周期。

```
int  EmaPeriod() const
```

返回值

返回指标创建时定义的 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# IndShift

返回指标的水平平移。

```
int  IndShift() const
```

返回值

返回指标创建时定义的水平平移值。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,          // 品名
   ENUM_TIMEFRAMES   period,          // 周期
   int               cmo_period,      // 动量周期
   int               ema_period,      // 平均周期
   int               ind_shift,       // 平移
   int               applied          // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*cmo_period*

[输入] 动量周期。

*ema_period*

[输入] 平均周期。

*ind_shift*

[输入] 水平平移。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_VIDYA 对应 CiVIDyA）。

© 2000-2025, MetaQuotes Ltd.

# 振荡器

这一组章节包含 MQL5 标准库内振荡器类的技术细节，以及它的所有关键组件的描述。

| 类/组 | 描述 |
| --- | --- |
| CiATR | 平均真实范围 |
| CiBearsPower | 空头力度 |
| CiBullsPower | 多头力度 |
| CiCCI | 商品通道指数 |
| CiChaikin | Chaikin 振荡器 |
| CiDeMarker | DeMarker |
| CiForce | 强力指数 |
| CiMACD | 移动均线聚合-离散 |
| CiMomentum | 动量 |
| CiOsMA | 移动均线振荡器（MACD 直方条） |
| CiRSI | 相对强度指数 |
| CiRVI | 相对动量指数 |
| CiStochastic | 随机振荡器 |
| CiWPR | 威廉姆斯百分比范围 |
| CiTriX | 三重指数移动均线振荡器 |

© 2000-2025, MetaQuotes Ltd.

# CiATR

CiATR 类旨在使用平均真实范围技术指标。

## 描述

CiATR 类可供创建，设置和访问平均真实范围指标的数据。

## 声明

```
class CiATR: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiATR

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_ATR 对应 CiATR)。

© 2000-2025, MetaQuotes Ltd.

# CiBearsPower

CiBearsPower类旨在使用空头力度技术指标。

## 描述

CiBearsPower类可供创建，设置和访问空头力度指标的数据。

## 声明

```
class CiBearsPower: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiBearsPower

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_BEARS 对应 CiBearsPower)。

© 2000-2025, MetaQuotes Ltd.

# CiBullsPower

CiBullsPower类旨在使用多头力度技术指标。

## 描述

CiBullsPower类可供创建，设置和访问多头力度指标的数据。

## 声明

```
class CiBullsPower: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiBullsPower

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_BULLS 对应 CiBullsPower)。

© 2000-2025, MetaQuotes Ltd.

# CiCCI

CiCCI类旨在使用商品通道指数技术指标。

## 描述

CiCCI类可供创建，设置和访问商品通道指数指标的数据。

## 声明

```
class CiCCI: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiCCI

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period,     // 平均周期
   int              applied        // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_CCI 对应 CiCCI)。

© 2000-2025, MetaQuotes Ltd.

# CiChaikin

CiChaikin 类旨在使用 Chaikin 振荡器技术指标。

## 描述

CiChaikin 类可供创建，设置和访问 Chaikin 振荡器指标的数据。

## 声明

```
class CiChaikin: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiChaikin

## 类方法

| 属性 | |
|---|---|
| FastMaPeriod | 返回快速 MA 线的平均周期 |
| SlowMaPeriod | 返回慢速 MA 线的平均周期 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# FastMaPeriod

返回快速 EMA 线的平均周期。

```
int  FastMaPeriod() const
```

返回值

返回指标创建时定义的快速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SlowMaPeriod

返回慢速 EMA 线的平均周期。

```
int  SlowMaPeriod() const
```

返回值

返回指标创建时定义的慢速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,              // 品名
   ENUM_TIMEFRAMES       period,              // 周期
   int                   fast_ma_period,      // 快速 EMA 周期
   int                   slow_ma_period,      // 慢速 EMA 周期
   ENUM_MA_METHOD        ma_method,           // 平均方法
   ENUM_APPLIED_VOLUME   applied              // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*fast_ma_period*

[输入] 快速 EMA 周期。

*slow_ma_period*

[输入] 慢速 EMA 周期。

*ma_method*

[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*

[输入] 交易量应用类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_CHAIKIN 对应 CiChaikin)。

© 2000-2025, MetaQuotes Ltd.

# CiDeMarker

CiDeMarker类旨在使用 DeMarker技术指标。

## 描述

CiDeMarker类可供创建，设置和访问 DeMarker指标的数据。

## 声明

```
class CiDeMarker: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiDeMarker

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_DEMARKER 对应 CiDeMarker)。

© 2000-2025, MetaQuotes Ltd.

# CiForce

CiForce 类旨在使用强力指数技术指标。

## 描述

CiForce 类可供创建，设置和访问强力指数指标的数据。

## 声明

```
class CiForce: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiForce

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,          // 品名
   ENUM_TIMEFRAMES       period,          // 周期
   int                   ma_period,       // 平均周期
   ENUM_MA_METHOD        ma_method,       // 平均方法
   ENUM_APPLIED_VOLUME   applied          // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*ma_method*

[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*

[输入] 交易量应用类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_FORCE 对应 CiForce)。

© 2000-2025, MetaQuotes Ltd.

# CiMACD

CiMACD 类旨在使用移动均线聚合-离散技术指标。

## 描述

CiMACD 类可供创建，设置和访问移动均线聚合-离散指标的数据。

## 声明

```
class CiMACD: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiMACD

## 类方法

| 属性 | |
|---|---|
| FastEmaPeriod | 返回快速 EMA 线的平均周期 |
| SlowEmaPeriod | 返回慢速 EMA 线的平均周期 |
| SignalPeriod | 返回信号线的平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回主线缓存区元素 |
| Signal | 返回信号线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# FastEmaPeriod

返回快速 EMA 线的平均周期。

```
int  FastEmaPeriod() const
```

返回值

返回指标创建时定义的快速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SlowEmaPeriod

返回慢速 EMA 线的平均周期。

```
int  SlowEmaPeriod() const
```

返回值

返回指标创建时定义的慢速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SignalPeriod

返回信号线的平均周期。

```
int  SignalPeriod() const
```

返回值

返回指标创建时定义的信号线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,              // 品名
   ENUM_TIMEFRAMES  period,              // 周期
   int              fast_ema_period,     // 快速 EMA 周期
   int              slow_ema_period,     // 慢速 EMA 周期
   int              signal_period,       // 信号周期
   int              applied              // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*fast_ema_period*

[输入] 快速 EMA 周期。

*slow_ema_period*

[输入] 慢速 EMA 周期。

*signal_period*

[输入] 信号线周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Signal

返回信号线指定索引处的缓存元素。

```
double  Signal(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

信号线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_MACD 对应 CiMACD)。

© 2000-2025, MetaQuotes Ltd.

# CiMomentum

CiMomentum 类旨在使用动量技术指标。

## 描述

CiMomentum 类可供创建，设置和访问动量指标的数据。

## 声明

```
class CiMomentum: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiMomentum

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period,     // 平均周期
   int              applied        // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_MOMENTUM 对应 CiMomentum）。

© 2000-2025, MetaQuotes Ltd.

# CiOsMA

CiOsMA 类旨在使用移动均线振荡器（MACD 直方条）技术指标。

## 描述

CiOsMA 类可供创建，设置和访问移动均线振荡器（MACD 直方条）指标的数据。

## 声明

```
class CiOsMA: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiOsMA

## 类方法

| 属性 | |
|---|---|
| FastEmaPeriod | 返回快速 EMA 线的平均周期 |
| SlowEmaPeriod | 返回慢速 EMA 线的平均周期 |
| SignalPeriod | 返回信号线的平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# FastEmaPeriod

返回快速 EMA 线的平均周期。

```
int  FastEmaPeriod() const
```

返回值

返回指标创建时定义的快速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SlowEmaPeriod

返回慢速 EMA 线的平均周期。

```
int  SlowEmaPeriod() const
```

返回值

返回指标创建时定义的慢速 EMA 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# SignalPeriod

返回信号线的平均周期。

```
int  SignalPeriod() const
```

返回值

返回指标创建时定义的信号线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,              // 品名
   ENUM_TIMEFRAMES  period,              // 周期
   int              fast_ema_period,     // 快速 EMA 周期
   int              slow_ema_period,     // 慢速 EMA 周期
   int              signal_period,       // 信号线周期
   int              applied              // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*fast_ema_period*

[输入] 快速 EMA 周期。

*slow_ema_period*

[输入] 慢速 EMA 周期。

*signal_period*

[输入] 信号线周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_OSMA 对应 CiOsMA）。

© 2000-2025, MetaQuotes Ltd.

# CiRSI

CiRSI类旨在使用相对强度指数技术指标。

## 描述

CiRSI类可供创建，设置和访问相对强度指数指标的数据。

## 声明

```
class CiRSI: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiRSI

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period,     // 平均周期
   int              applied        // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_RSI 对应 CiRSI)。

© 2000-2025, MetaQuotes Ltd.

# CiRVI

CiRVI类旨在使用相对动量指数技术指标。

## 描述

CiRVI类可供创建，设置和访问相对动量指数指标的数据。

## 声明

```
class CiRVI: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiRVI

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回基准线缓存区元素 |
| Signal | 返回信号线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period      // 平均周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Signal

返回信号线指定索引处的缓存元素。

```
double  Signal(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

信号线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_RVI 对应 CiRVI）。

© 2000-2025, MetaQuotes Ltd.

# CiStochastic

CiStochastic 类旨在使用随机振荡器技术指标。

## 描述

CiStochastic 类可供创建，设置和访问随机振荡器指标的数据。

## 声明

```
class CiStochastic: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiStochastic

## 类方法

| 属性 | |
|---|---|
| Kperiod | 返回 %K 线的平均周期 |
| Dperiod | 返回 %D 线的平均周期 |
| Slowing | 返回慢速周期 |
| MaMethod | 返回平均方法 |
| PriceField | 应用的价格类型（最低价/最高价 或 收盘价/收盘价） |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回基准线缓存区元素 |
| Signal | 返回信号线缓存区元素 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# Kperiod

返回 %K 线的平均周期。

```
int  Kperiod() const
```

返回值

返回指标创建时定义的 %K 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Dperiod

返回 %D 线的平均周期。

```
int  Dperiod() const
```

返回值

返回指标创建时定义的 %D 线的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Slowing

返回慢速周期。

```
int  Slowing() const
```

返回值

返回指标创建时定义的慢速周期。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# PriceField

返回应用价格类型（最低价/最高价 或 收盘价/收盘价）。

```
ENUM_STO_PRICE  PriceField() const
```

返回值

在指标创建时定义的应用价格类型。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,          // 品名
   ENUM_TIMEFRAMES   period,          // 周期
   int               Kperiod,         // 平均周期 %K
   int               Dperiod,         // 平均周期 %D
   int               slowing,         // 慢速周期
   ENUM_MA_METHOD    ma_method,       // 平均方法
   ENUM_STO_PRICE    price_field      // 应用价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*Kperiod*

[输入] %K 线的平均周期。

*Dperiod*

[输入] %D 线的平均周期。

*slowing*

[输入] 慢速周期。

*ma_method*

[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*price_field*

[输入] 应用的价格类型 (最低价/最高价 或 收盘价/收盘价) (ENUM_STO_PRICE 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

## Signal

返回信号线指定索引处的缓存元素。

```
double  Signal(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

## Signal

返回信号线指定索引处的缓存元素。

```
double  Signal(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

信号线指定索引处的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_STOCHASTIC 对应 CiStochastic)。

© 2000-2025, MetaQuotes Ltd.

# CiTriX

CiTriX 类旨在使用三重指数移动均线振荡器技术指标。

## 描述

CiTriX 类可供创建，设置和访问三重指数移动均线振荡器指标的数据。

## 声明

```
class CiTriX: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiTriX

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,        // 品名
   ENUM_TIMEFRAMES  period,        // 周期
   int              ma_period,     // 平均周期
   int              applied        // 处理或应用的价格类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*applied*

[输入] 处理或应用的价格类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_TRIX 对应 CiTriX)。

© 2000-2025, MetaQuotes Ltd.

# CiWPR

CiWPR 类旨在使用威廉姆斯百分比范围技术指标。

## 描述

CiWPR 类可供创建，设置和访问威廉姆斯百分比范围指标的数据。

## 声明

```
class CiWPR: public CIndicator
```

## 标称库文件

```
#include <Indicators\Oscilators.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiWPR

## 类方法

| 属性 | |
|---|---|
| CalcPeriod | 返回计算周期 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# CalcPeriod

返回计算周期。

```
int  CalcPeriod() const
```

返回值

返回指标创建时定义的计算周期。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,          // 品名
   ENUM_TIMEFRAMES  period,          // 周期
   int              calc_period      // 计算周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*calc_period*

[输入] 计算周期。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_WPR 对应 CiWPR)。

© 2000-2025, MetaQuotes Ltd.

# 成交量指标

这一组章节包含 MQL5 标准库中成交量指标类的技术细节，以及它的所有关键组件的描述。

| 类/组 | 描述 |
| --- | --- |
| CiAD | 累积/派发 |
| CiMFI | 资金流指数 |
| CiOBV | 平衡成交量 |
| CiVolumes | 成交量 |

© 2000-2025, MetaQuotes Ltd.

# CiAD

CiAD 类旨在使用累积/派发技术指标。

## 描述

CiAD 类可供创建，设置和访问累积/派发指标的数据。

## 声明

```
class CiAD: public CIndicator
```

## 标称库文件

```
#include <Indicators\Volumes.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiAD

## 类方法

| 属性 | |
|---|---|
| Applied | 返回交易量应用类型。 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,      // 品名
   ENUM_TIMEFRAMES       period,      // 周期
   ENUM_APPLIED_VOLUME   applied      // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*applied*

[输入] 应用的成交量类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_AD 对应 CiAD)。

© 2000-2025, MetaQuotes Ltd.

# CiMFI

CiMFI类旨在使用资金流指数技术指标。

## 描述

CiMFI类可供创建，设置和访问资金流指数指标的数据。

## 声明

```
class CiMFI: public CIndicator
```

## 标称库文件

```
#include <Indicators\Volumes.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiMFI

## 类方法

| 属性 | |
|---|---|
| MaPeriod | 返回平均周期 |
| Applied | 返回交易量应用类型。 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# MaPeriod

返回平均周期。

```
int  MaPeriod() const
```

返回值

返回指标创建时定义的平均周期。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,          // 品名
   ENUM_TIMEFRAMES       period,          // 周期
   int                   ma_period,       // 平均周期
   ENUM_APPLIED_VOLUME   applied          // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*ma_period*

[输入] 平均周期。

*applied*

[输入] 应用的成交量类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_MFI 对应 CiMFI)。

© 2000-2025, MetaQuotes Ltd.

# CiOBV

CiOBV 类旨在使用平衡成交量技术指标。

## 描述

CiOBV 类可供创建，设置和访问平衡成交量指标的数据。

## 声明

```
class CiOBV: public CIndicator
```

## 标称库文件

```
#include <Indicators\Volumes.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiOBV

## 类方法

| 属性 | |
|---|---|
| Applied | 返回交易量应用类型。 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,      // 品名
   ENUM_TIMEFRAMES       period,      // 周期
   ENUM_APPLIED_VOLUME   applied      // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*applied*

[输入] 应用的成交量类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_OBV 对应 CiOBV)。

© 2000-2025, MetaQuotes Ltd.

# CiVolumes

CiVolumes 类旨在使用成交量技术指标。

## 描述

CiVolumes 类可供创建，设置和访问成交量指标的数据。

## 声明

```
class CiVolumes: public CIndicator
```

## 标称库文件

```
#include <Indicators\Volumes.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiVolumes

## 类方法

| 属性 | |
|---|---|
| Applied | 返回交易量应用类型。 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,      // 品名
   ENUM_TIMEFRAMES       period,      // 周期
   ENUM_APPLIED_VOLUME   applied      // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*applied*

[输入] 应用的成交量类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_VOLUMES 对应 CiVolumes)。

© 2000-2025, MetaQuotes Ltd.

# 比尔威廉姆斯指标

这一组章节包含 MQL5 标准库中比尔威廉姆斯指标类的技术细节，以及它的所有关键组件的描述。

| 类/组 | 描述 |
| --- | --- |
| CiAC | 加速计振荡器 |
| CiAlligator | 鳄鱼 |
| CiAO | 动量震荡器 |
| CiFractals | 分形 |
| CiGator | 鳄鱼振荡器 |
| CiBWMFI | 市场促进指数 |

© 2000-2025, MetaQuotes Ltd.

# CiAC

CiAC 类旨在使用加速计振荡器技术指标。

## 描述

CiAC 类可供创建，设置和访问加速计振荡器指标的数据。

## 声明

```
class CiAC: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiAC

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_AC 对应 CiAC）。

© 2000-2025, MetaQuotes Ltd.

# CiAlligator

CiAlligator类旨在使用鳄鱼技术指标。

## 描述

CiAlligator类可供创建，设置和访问鳄鱼指标的数据。

## 声明

```
class CiAlligator: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiAlligator

## 类方法

| 属性 | |
|---|---|
| JawPeriod | 返回下颌线的平均周期 |
| JawShift | 返回下颌线的水平平移 |
| TeethPeriod | 返回齿线的平均周期 |
| TeethShift | 返回齿线的水平平移 |
| LipsPeriod | 返回唇线的平均周期 |
| LipsShift | 返回唇线的水平平移 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Jaw | 返回下颌线缓存区的缓存数据 |
| Teeth | 返回齿线缓存区的缓存数据 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| Lips | 返回唇线缓存区的缓存数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# JawPeriod

返回下颌线的平均周期。

```
int  JawPeriod() const
```

返回值

返回下颌线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# JawShift

返回下颌线的水平平移。

```
int  JawShift() const
```

返回值

下颌线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# TeethPeriod

返回齿线的平均周期。

```
int  TeethPeriod() const
```

返回值

返回齿线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# TeethShift

返回齿线的水平平移。

```
int  TeethShift() const
```

返回值

齿线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# LipsPeriod

返回唇线的平均周期。

```
int  LipsPeriod() const
```

返回值

返回唇线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# LipsShift

返回唇线的水平平移。

```
int  LipsShift() const
```

返回值

唇线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,           // 品名
   ENUM_TIMEFRAMES   period,           // 周期
   int               jaw_period,       // 下颌周期
   int               jaw_shift,        // 下颌平移
   int               teeth_period,     // 齿周期
   int               teeth_shift,      // 齿平移
   int               lips_period,      // 唇周期
   int               lips_shift,       // 唇平移
   ENUM_MA_METHOD    ma_method,        // 平均方法
   int               applied           // 交易量应用类型
   )
```

参数

*symbol*
[输入] 品名。

*period*
[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*jaw_period*
[输入] 下颌周期。

*jaw_shift*
[输入] 下颌平移。

*teeth_period*
[输入] 齿周期。

*teeth_shift*
[输入] 齿平移。

*lips_period*
[输入] 唇周期。

*lips_shift*
[输入] 唇平移。

*ma_method*
[输入] 移动均线方法 (ENUM_MA_METHOD 枚举)。

*applied*
[输入] 交易量应用类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Jaw

返回下颌线指定索引的缓存元素。。

```
double  Jaw(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

下颌线指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Teeth

返回齿线指定索引的缓存元素。

```
double  Teeth(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

齿线指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Lips

返回唇线指定索引的缓存元素。

```
double  Lips(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

唇线指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_ALLIGATOR 对应 CiAlligator)。

© 2000-2025, MetaQuotes Ltd.

# CiAO

CiAO 类旨在使用动量振荡器技术指标。

## 描述

CiAO 类可供创建，设置和访问动量振荡器指标的数据。

## 声明

```
class CiAO: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiAO

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_AO 对应 CiAO）。

© 2000-2025, MetaQuotes Ltd.

# CiFractals

CiFractals 类旨在使用分形技术指标。

## 描述

CiFractals 类可供创建，设置和访问分形指标的数据。

## 声明

```
class CiFractals: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiFractals

## 类方法

| 创建方法 | |
|---|---|
| 创建 | 创建指标 |
| 数据访问方法 | |
| Upper | 返回上部缓存区的缓存数据 |
| Lower | 返回下部缓存区的缓存数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类 CObject

Prev, Prev, Next, Next, Compare

方法继承自类 CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类 CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string           symbol,     // 品名
   ENUM_TIMEFRAMES  period      // 周期
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

true 如果成功, false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Upper

返回上部缓存区指定索引的缓存元素。

```
double  Upper(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

上部缓存区指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Lower

返回下部缓存区指定索引的缓存元素。

```
double  Lower(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

下部缓存区指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_FRACTALS 对应 CiFractals)。

© 2000-2025, MetaQuotes Ltd.

# CiGator

CiGator类旨在使用鳄鱼振荡器技术指标。

## 描述

CiGator类可供创建，设置和访问鳄鱼振荡器指标的数据。

## 声明

```
class CiGator: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
CArray
CArrayObj
CSeries
CIndicator
CiGator

## 类方法

| 属性 | |
|---|---|
| JawPeriod | 返回下颌线的平均周期 |
| JawShift | 返回下颌线的水平平移 |
| TeethPeriod | 返回齿线的平均周期 |
| TeethShift | 返回齿线的水平平移 |
| LipsPeriod | 返回唇线的平均周期 |
| LipsShift | 返回唇线的水平平移 |
| MaMethod | 返回平均方法 |
| Applied | 返回处理或应用的价格类型 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Upper | 返回上部缓存区的缓存数据 |
| Lower | 返回下部缓存区的缓存数据 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# JawPeriod

返回下颌线的平均周期。

```
int  JawPeriod() const
```

返回值

返回下颌线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# JawShift

返回下颌线的水平平移。

```
int  JawShift() const
```

返回值

下颌线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# TeethPeriod

返回齿线的平均周期。

```
int  TeethPeriod() const
```

返回值

返回齿线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# TeethShift

返回齿线的水平平移。

```
int  TeethShift() const
```

返回值

齿线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# LipsPeriod

返回唇线的平均周期。

```
int  LipsPeriod() const
```

返回值

返回唇线的平均周期，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# LipsShift

返回唇线的水平平移。

```
int  LipsShift() const
```

返回值

唇线的水平平移，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# MaMethod

返回平均方法。

```
ENUM_MA_METHOD  MaMethod() const
```

返回值

返回平均方法，指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# Applied

返回处理或应用的价格类型。

```
int  Applied() const
```

返回值

处理或应用的 价格类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string            symbol,           // 品名
   ENUM_TIMEFRAMES   period,           // 周期
   int               jaw_period,       // 下颌周期
   int               jaw_shift,        // 下颌平移
   int               teeth_period,     // 齿周期
   int               teeth_shift,      // 齿平移
   int               lips_period,      // 唇周期
   int               lips_shift,       // 唇平移
   ENUM_MA_METHOD    ma_method,        // 平均方法
   int               applied           // 交易量应用类型
   )
```

参数

*symbol*
[输入] 品名。

*period*
[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*jaw_period*
[输入] 下颌周期。

*jaw_shift*
[输入] 下颌平移。

*teeth_period*
[输入] 齿周期。

*teeth_shift*
[输入] 齿平移。

*lips_period*
[输入] 唇周期。

*lips_shift*
[输入] 唇平移。

*ma_method*
[输入] 平均方法 (ENUM_MA_METHOD 枚举)。

*applied*
[输入] 交易量应用类型。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Upper

返回上部缓存区指定索引的缓存元素。

```
double  Upper(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

上部缓存区指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Lower

返回下部缓存区指定索引的缓存元素。

```
double  Upper(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

下部缓存区指定索引的缓存元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_GATOR 对应 CiGator)。

© 2000-2025, MetaQuotes Ltd.

# CiBWMFI

CiBWMFI类旨在使用比尔·威廉姆斯市场促进指数技术指标。

## 描述

CiBWMFI类可供创建，设置和访问比尔·威廉姆斯市场促进指数指标的数据。

## 声明

```
class CiBWMFI: public CIndicator
```

## 标称库文件

```
#include <Indicators\BillWilliams.mqh>
```

## 继承体系

CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiBWMFI

## 类方法

| 属性 | |
|---|---|
| Applied | 返回交易量应用类型。 |
| 创建方法 | |
| 创建 | 创建指标 |
| 数据访问方法 | |
| Main | 返回缓存区数据 |
| 输入/输出 | |
| virtual Type | 返回对象类型的标识符 |

方法继承自类CObject

Prev, Prev, Next, Next, Compare

方法继承自类CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

方法继承自类CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

方法继承自类CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

方法继承自类CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

© 2000-2025, MetaQuotes Ltd.

# Applied

返回交易量应用类型。

```
ENUM_APPLIED_VOLUME  Applied() const
```

返回值

交易量应用类型，在指标创建时定义。

© 2000-2025, MetaQuotes Ltd.

# 创建

它以指定参数创建指标。使用 Refresh() 和 GetData() 来更新和获取指标数据。

```
bool  Create(
   string                symbol,      // 品名
   ENUM_TIMEFRAMES       period,      // 周期
   ENUM_APPLIED_VOLUME   applied      // 交易量应用类型
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*applied*

[输入] 交易量应用类型 (ENUM_APPLIED_VOLUME 枚举)。

返回值

true 如果成功，false 如果指标未能创建。

© 2000-2025, MetaQuotes Ltd.

# Main

返回缓存区指定索引的元素。

```
double  Main(
   int  index      // 索引
   )
```

参数

*index*

[输入] 元素索引。

返回值

如果成功是缓存区指定索引的元素，或 EMPTY_VALUE 如果此处没有任何正确数据。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符 (IND_BWMFI 对应 CiBWMFI)。

© 2000-2025, MetaQuotes Ltd.

# CiCustom

CiCustom 类旨在使用自定义技术指标。

## 描述

CiCustom 类可供创建，设置和访问自定义指标的数据。

## 声明

```
class CiCustom: public CIndicator
```

## 标称库文件

```
#include <Indicators\Custom.mqh>
```

## 类方法

| 属性 | |
|---|---|
| NumBuffers | 设置缓存区数量 |
| NumParams | 获取参数数量 |
| ParamType | 获取指定参数类型 |
| ParamLong | 获取指定整数类型参数的值 |
| ParamDouble | 获取指定双精度类型参数的值 |
| ParamString | 获取指定字符串类型参数的值 |
| 输入/输出 | |
| virtual Type | 获取对象类型的标识符 |

© 2000-2025, MetaQuotes Ltd.

# NumBuffers

设置缓存区数量。

```
bool  NumBuffers(
   int  buffers      // 缓存区数量
   )
```

返回值

true 如果成功，false 如果缓存区未能设置。

© 2000-2025, MetaQuotes Ltd.

# NumParams

获取参数数量。

```
int  NumParams() const
```

返回值

创建指标时使用的参数数量。

© 2000-2025, MetaQuotes Ltd.

# ParamType

获取指定索引处参数的类型。

```
ENUM_DATATYPE  ParamType(
   int  index      // 参数索引
   ) const
```

参数

*index*

[输入] 参数索引。

返回值

返回指标创建时使用的指定索引处的参数类型 (值为 ENUM_DATATYPE 枚举)。

注释

如果参数索引无效，它返回 WRONG_VALUE。

© 2000-2025, MetaQuotes Ltd.

# ParamLong

获取指定长整数类型参数的值。

```
long  ParamLong(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 参数索引。

返回值

指定指标创建时使用的长整数类型参数的值。

注释

如果该值无效，它返回 0。

© 2000-2025, MetaQuotes Ltd.

# ParamDouble

获取指定双精度类型参数的值。

```
double  ParamDouble(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 参数索引。

返回值

指定指标创建时使用的双精度类型参数的值。

注释

如果该值无效，它返回 EMPTY_VALUE。

© 2000-2025, MetaQuotes Ltd.

# ParamString

获取指定字符串类型参数的值。

```
string  ParamString(
   int  index      // 索引
   ) const
```

参数

*index*

[输入] 参数索引。

返回值

指定指标创建时使用的字符串类型参数的值。

注释

如果该值无效，它返回空字串。

© 2000-2025, MetaQuotes Ltd.

# Type

返回对象类型的标识符。

```
virtual int  Type() const
```

返回值

对象类型的标识符（IND_CUSTOM 对应 CiCustom）。

© 2000-2025, MetaQuotes Ltd.

# 交易类

这章节包含操作交易类的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建自程序（脚本，智能程序）时，使用交易类，将节省大量时间。

MQL5 标准库（就数据集而言）位于终端工作目录的 Include\Arrays 文件夹。

| 类/组 | 描述 |
| --- | --- |
| CAccountInfo | 用于操作交易账户属性的类 |
| CSymbolInfo | 用于操作交易工具属性的类 |
| COrderInfo | 用于操作挂单属性的类 |
| CHistoryOrderInfo | 用于操作历史订单属性的类 |
| CPositionInfo | 用于操作仓位属性的类 |
| CDealInfo | 用于操作历史成交属性的类 |
| CTrade | 用于交易操作执行的类 |
| CTerminalInfo | 获取 mql5 程序环境属性的类 |

© 2000-2025, MetaQuotes Ltd.

# CAccountInfo

CAccountInfo 类用于简便地访问当前已创建的交易账户属性。

## 描述

CAccountInfo 类用于简便地访问当前已创建的交易账户属性。

## 声明

```
class CAccountInfo : public CObject
```

## 标称库文件

```
#include <Trade\AccountInfo.mqh>
```

## 继承体系

CObject

CAccountInfo

## 类方法分组

| 访问整数型属性 | |
|---|---|
| Login | 获取账户号码 |
| TradeMode | 获取交易模式 |
| TradeModeDescription | 获取交易模式的字符串描述 |
| Leverage | 获取给定的杠杆额度 |
| StopoutMode | 获取账户爆仓模式 |
| StopoutModeDescription | 获取账户爆仓模式的描述 |
| TradeAllowed | 获取交易许可标志 |
| TradeExpert | 获取自动交易许可标志 |
| LimitOrders | 获取许可的最大挂单数量 |
| MarginMode | 获取预付款计算模式 |
| MarginModeDescription | 获取预付款计算模式为字符串 |
| 访问双精度型属性 | |
| Balance | 获取账户结余 |
| Credit | 获取账户给定的信用额度 |
| Profit | 获取账户的当前盈利额度 |
| Equity | 获取账户当前净值额度 |

© 2000-2025, MetaQuotes Ltd.

| 访问整数型属性 | |
|---|---|
| Margin | 获取保留保证金的额度 |
| FreeMargin | 获取可用保证金额度 |
| MarginLevel | 获取保证金比例 |
| MarginCall | 获取入金的保证金比例 |
| MarginStopOut | 获取爆仓的保证金比例 |
| 访问文本型属性 | |
| 名称 | 获取客户名称 |
| Server | 获取交易服务器名 |
| Currency | 获取入金货币名 |
| Company | 获取开户券商公司名称 |
| 访问 MQL5 API 函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值 |
| 附加方法 | |
| OrderProfitCheck | 获取基于传递参数的评估盈利 |
| MarginCheck | 获取执行交易操作所需的保证金额度 |
| FreeMarginCheck | 获取执行交易操作之后剩余的可用保证金额度 |
| MaxLotCheck | 获取交易操作的最大可能交易量 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Login

获取账户号码。

```
long  Login() const
```

返回值

账户号码。

© 2000-2025, MetaQuotes Ltd.

# TradeMode

获取交易模式。

```
ENUM_ACCOUNT_TRADE_MODE  TradeMode() const
```

返回值

交易模式（值为 ENUM_ACCOUNT_TRADE_MODE 枚举）。

© 2000-2025, MetaQuotes Ltd.

# TradeModeDescription

获取交易模式的字符串描述。

```
string  TradeModeDescription() const
```

返回值

交易模式的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# Leverage

获取给定的杠杆额度。

```
long  Leverage() const
```

返回值

给定的杠杆额度。

© 2000-2025, MetaQuotes Ltd.

# StopoutMode

获取账户爆仓模式。

```
ENUM_ACCOUNT_STOPOUT_MODE  StopoutMode() const
```

返回值

账户爆仓模式（值为 ENUM_ACCOUNT_STOPOUT_MODE 枚举）。

© 2000-2025, MetaQuotes Ltd.

# StopoutModeDescription

获取设置最小保证金比例模式的字符串描述。

```
string  StopoutModeDescription() const
```

返回值

最小保证金比例模式的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# MarginMode

获取预付款计算模式。

```
ENUM_ACCOUNT_MARGIN_MODE  MarginMode() const
```

返回值

从ENUM_ACCOUNT_MARGIN_MODE枚举的预付款计算模式。

© 2000-2025, MetaQuotes Ltd.

# MarginModeDescription

获取预付款计算模式为字符串。

```
string  MarginModeDescription() const
```

返回值

预付款计算模式为字符串。

© 2000-2025, MetaQuotes Ltd.

# TradeAllowed

获取交易许可标志。

```
bool  TradeAllowed() const
```

返回值

交易许可标志。

© 2000-2025, MetaQuotes Ltd.

# TradeExpert

获取自动交易许可标志。

```
bool  TradeExpert() const
```

返回值

自动交易许可标志。

© 2000-2025, MetaQuotes Ltd.

# LimitOrders

获取许可的最大挂单数量

```
int  LimitOrders() const
```

返回值

许可的最大挂单数量。

注释

0 - 无限制。

© 2000-2025, MetaQuotes Ltd.

# Balance

获取账户结余。

```
double  Balance() const
```

返回值

账户结余（以入金货币为单位）。

© 2000-2025, MetaQuotes Ltd.

# Credit

获取账户给定的信用额度。

```
double  Credit() const
```

返回值

给定的信用额度 (以入金货币为单位)。

© 2000-2025, MetaQuotes Ltd.

# Profit

获取账户的当前盈利额度。

```
double  Profit() const
```

返回值

账户当前盈利额度 (以入金货币为单位)。

© 2000-2025, MetaQuotes Ltd.

# Equity

获取账户当前净值额度。

```
double  Equity() const
```

返回值

账户当前净值额度 (以入金货币为单位)。

© 2000-2025, MetaQuotes Ltd.

# Margin

获取保留保证金的额度。

```
double  Margin() const
```

返回值

保留保证金的额度 (以入金货币为单位)。

© 2000-2025, MetaQuotes Ltd.

# FreeMargin

获取可用保证金额度。

```
double  FreeMargin() const
```

返回值

可用保证金额度 (以入金货币为单位)。

© 2000-2025, MetaQuotes Ltd.

# MarginLevel

获取保证金比例。

```
double  MarginLevel() const
```

返回值

保证金比例。

© 2000-2025, MetaQuotes Ltd.

# MarginCall

获取入金的保证金比例。

```
double  MarginCall() const
```

返回值

入金的保证金比例。

© 2000-2025, MetaQuotes Ltd.

# MarginStopOut

获取爆仓的保证金比例。

```
double  MarginStopOut() const
```

返回值

返回 爆仓的保证金比例。

© 2000-2025, MetaQuotes Ltd.

# 名称

获取客户名称。

```
string  Name() const
```

返回值

客户名称。

© 2000-2025, MetaQuotes Ltd.

# Server

获取交易服务器名。

```
string  Server() const
```

返回值

交易服务器名。

© 2000-2025, MetaQuotes Ltd.

# Currency

获取入金货币名。

```
string  Currency() const
```

返回值

入金货币名。

© 2000-2025, MetaQuotes Ltd.

# Company

获取开户券商公司名称。

```
string  Company() const
```

返回值

开户券商公司名称。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
long  InfoInteger(
   ENUM_ACCOUNT_INFO_INTEGER  prop_id     // 属性 ID
   ) const
```

参数

*prop_id*

[输入] 属性标识符。其值可为 ENUM_ACCOUNT_INFO_INTEGER 枚举之一。

返回值

返回 长整数 型数值。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
double  InfoDouble(
   ENUM_ACCOUNT_INFO_DOUBLE  prop_id     // 属性 ID
   ) const
```

参数

*prop_id*

[输入] 属性标识符。其值可为 ENUM_ACCOUNT_INFO_DOUBLE 枚举之一。

返回值

返回 双精度 型的数值。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
string  InfoString(
   ENUM_ACCOUNT_INFO_STRING  prop_id     // 属性 ID
   ) const
```

参数

*prop_id*

[输入] 属性标识符。其值可为 ENUM_ACCOUNT_INFO_STRING 枚举之一。

返回值

返回 字符串 型的值。

© 2000-2025, MetaQuotes Ltd.

# OrderProfitCheck

函数检查基于的传递参数计算当前账户的盈利。函数用于某个交易操作结果的预估。返回的数值以账户货币为单位。

```
double  OrderProfitCheck(
   const string        symbol,              // 品名
   ENUM_ORDER_TYPE     trade_operation,     // 操作类型 (ORDER_TYPE_BUY or ORDER_TYPE_S
   double              volume,              // 交易量
   double              price_open,          // 开仓价
   double              price_close          // 平仓价
   ) const
```

参数

*symbol*

[输入] 交易操作的品名。

*trade_operation*

[输入] 交易操作类型 (ENUM_ORDER_TYPE 枚举)。

*volume*

[输入] 交易操作类型。

*price_open*

[输入] 开仓价。

*price_close*

[输入] 平仓价。

返回值

如果成功，它返回盈利额度或在出错情况下 EMPTY_VALUE。

© 2000-2025, MetaQuotes Ltd.

# MarginCheck

获取执行交易操作所需的保证金额度。

```
double  MarginCheck(
   const string         symbol,              // 品名
   ENUM_ORDER_TYPE      trade_operation,     // 操作
   double               volume,              // 交易量
   double               price                // 价格
   ) const
```

参数

*symbol*

[输入] 交易操作的品名。

*trade_operation*

[输入] 交易操作类型 (ENUM_ORDER_TYPE 枚举)。

*volume*

[输入] 交易操作类型。

*price*

[输入] 交易操作价格。

返回值

执行交易操作所需的保证金额度。

© 2000-2025, MetaQuotes Ltd.

# FreeMarginCheck

获取执行交易操作之后剩余的可用保证金额度。

```
double  FreeMarginCheck(
   const string         symbol,              // 品名
   ENUM_ORDER_TYPE      trade_operation,     // 操作
   double               volume,              // 交易量
   double               price                // 价格
   ) const
```

参数

*symbol*

[输入] 交易操作的品名。

*trade_operation*

[输入] 交易操作类型 (ENUM_ORDER_TYPE 枚举)。

*volume*

[输入] 交易操作类型。

*price*

[输入] 交易操作价格。

返回值

执行交易操作之后剩余的可用保证金额度。

© 2000-2025, MetaQuotes Ltd.

# MaxLotCheck

获取交易操作的最大可能交易量。

```
double  MaxLotCheck(
   const string         symbol,              // 品名
   ENUM_ORDER_TYPE      trade_operation,     // 操作
   double               price,               // 价格
   double               percent=100          // 用于交易操作的可用保证金百分比 (0-100%)
   ) const
```

参数

*symbol*

[输入] 交易操作的品名。

*trade_operation*

[输入] 交易操作类型 (ENUM_ORDER_TYPE 枚举)。

*price*

[输入] 交易操作价格。

*percent=100*

[输入] 用于交易操作的可用保证金百分比 (0-100%)。

返回值

交易操作的最大可能交易量。

© 2000-2025, MetaQuotes Ltd.

# CSymbolInfo

CSymbolInfo 类用于简便地访问品种属性。

## 描述

CSymbolInfo 类可供访问品种属性。

## 声明

```
class CSymbolInfo : public CObject
```

## 标称库文件

```
#include <Trade\SymbolInfo.mqh>
```

## 继承体系

CObject

CSymbolInfo

## 类方法分组

| | |
|---|---|
| 控制 | |
| Refresh | 刷新品种数据 |
| RefreshRates | 刷新品种报价 |
| 属性 | |
| 名称 | 获取/设置品种名称 |
| Select | 获取/设置"市场观察"品种标志 |
| IsSynchronized | 检查品种与服务器之间的同步 |
| 成交量 | |
| Volume | 获取最后一笔成交单的成交量 |
| VolumeHigh | 获取日内最大交易量 |
| VolumeLow | 获取日内最小交易量 |
| 综合 | |
| Time | 获取最后一笔报价时间 |
| Spread | 获取点差（以点数为单位） |
| SpreadFloat | 获取浮动点差标志 |
| TicksBookDepth | 获取即时报价保存的深度 |
| 级别 | |

© 2000-2025, MetaQuotes Ltd.

| 控制 | |
|---|---|
| StopsLevel | 获取订单距现价的最小距离（以点数为单位） |
| FreezeLevel | 获取交易操作的冻结距离（以点数为单位） |
| 卖出价格 | |
| Bid | 获取当前卖出价格 |
| BidHigh | 获取日内最高卖出价格 |
| BidLow | 获取日内最低卖出价格 |
| 买入价格 | |
| Ask | 获取当前买入价格 |
| AskHigh | 获取日内最高买入价格 |
| AskLow | 获取日内最低买入价格 |
| 价格 | |
| Last | 获取当前最后价格 |
| LastHigh | 获取日内最高最后价格 |
| LastLow | 获取日内最低最后价格 |
| 交易模式 | |
| TradeCalcMode | 获取合约成本计算模式 |
| TradeCalcModeDescription | 获取合约成本计算模式的字符串描述 |
| TradeMode | 获取订单执行类型 |
| TradeModeDescription | 获取订单执行类型的字符串描述 |
| TradeExecution | 获取交易执行模式 |
| TradeExecutionDescription | 获取交易执行模式的字符串描述 |
| 掉期利率 | |
| SwapMode | 获取掉期利率的计算模式 |
| SwapModeDescription | 获取掉期利率的计算模式的字符串描述 |
| SwapRollover3days | 获取三日掉期利率费用 |
| SwapRollover3daysDescription | 获取三日掉期利率费用的字符串描述 |
| 保证金和标志 | |
| MarginInitial | 获取初始保证金值 |
| MarginMaintenance | 获取维护保证金值 |
| MarginLong | 获取支付多头仓位的保证金率 |

© 2000-2025, MetaQuotes Ltd.

| 控制 | |
|---|---|
| MarginShort | 获取支付空头仓位的保证金率 |
| MarginLimit | 获取支付限价单的保证金率 |
| MarginStop | 获取支付突破单的保证金率 |
| MarginStopLimit | 获取支付双向挂单的保证金率 |
| TradeTimeFlags | 获取允许的订单过期模式标志 |
| TradeFillFlags | 获取允许的订单填充模式标志 |
| 量化 | |
| Digits | 获取小数点之后的位数 |
| Point | 获取点数值 |
| TickValue | 获取即时报价成本（最小价格变化） |
| TickValueProfit | 获取经计算的盈利仓位的即时价格 |
| TickValueLoss | 获取经计算的亏损仓位的即时价格 |
| TickSize | 获取价格的最小变化 |
| 合约大小 | |
| ContractSize | 获取交易合约的额度 |
| LotsMin | 获取一笔成交的最小成交量 |
| LotsMax | 获取一笔成交的最大成交量 |
| LotsStep | 获取一笔成交的最小变化步长 |
| LotsLimit | 获取一个品种的最大允许开仓和挂单交易量（方向无关） |
| 掉期利率大小 | |
| SwapLong | 获取多头持仓的掉期利率值 |
| SwapShort | 获取空头持仓的掉期利率值 |
| 文本属性 | |
| CurrencyBase | 获取品名的基准货币 |
| CurrencyProfit | 获取盈利货币名 |
| CurrencyMargin | 获取保证金货币名 |
| Bank | 获取当前报价来源的名称 |
| 描述 | 获取品种的字符串描述 |
| Path | 获取品种树的路径 |
| 品种属性 | |

© 2000-2025, MetaQuotes Ltd.

| 控制 | |
|---|---|
| SessionDeals | 获取当前时段的成交单数量 |
| SessionBuyOrders | 获取当前时刻的多头订单数量 |
| SessionSellOrders | 获取当前时刻的空头订单数量 |
| SessionTurnover | 获取当前时段的流水摘要 |
| SessionInterest | 获取当前时段的持仓摘要 |
| SessionBuyOrdersVolume | 获取多头订单的当前交易量 |
| SessionSellOrdersVolume | 获取空头订单的当前交易量 |
| SessionOpen | 获取当前时段的开仓价 |
| SessionClose | 获取当前时段的平仓价 |
| SessionAW | 获取当前时段的平均加权价 |
| SessionPriceSettlement | 获取当前时段的结算价 |
| SessionPriceLimitMin | 获取当前时段的最小价 |
| SessionPriceLimitMax | 获取当前时段的最大价 |
| 访问 MQL5 API 函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值。 |
| 服务函数 | |
| NormalizePrice | 返回品种属性的规整化价格数值 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Refresh

刷新品种数据。

```
void  Refresh()
```

返回值

无。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# RefreshRates

刷新品种报价数据。

```
bool  RefreshRates()
```

返回值

true -成功情况下 false -如果不能刷新报价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

## 名称

获取品种名称。

```
string  Name() const
```

返回值

品种名称。

## 名称

设置品种名称。

```
bool  Name(string name)
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Select

获取"市场观察"品种标志。

```
bool  Select() const
```

返回值

获取"市场观察"品种标志。

## Select

设置"市场观察"品种标志。

```
bool  Select()
```

返回值

true -成功情况下 false -如果不能改变标志。

© 2000-2025, MetaQuotes Ltd.

# IsSynchronized

检查品种与服务器之间的同步。

```
bool  IsSynchronized() const
```

返回值

true -如果品种与服务器已经同步，false -如果没有的话。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Volume

获取最后一笔成交单的成交量。

```
long  Volume() const
```

返回值

最后一笔成交单的成交量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeHigh

获取日内最大成交量。

```
long  VolumeHigh() const
```

返回值

日内最大成交量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeLow

获取日内最小成交量。

```
long  VolumeLow() const
```

返回值

日内最小成交量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Time

获取最后一笔报价时间。

```
datetime  Time() const
```

返回值

最后一笔报价时间。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Spread

获取点差（以点数为单位）。

```
int  Spread() const
```

返回值

获取点差（以点数为单位）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SpreadFloat

获取浮动点差标志。

```
bool  SpreadFloat() const
```

返回值

浮动点差标志。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TicksBookDepth

获取即时报价保存的深度。

```
int  TicksBookDepth() const
```

返回值

即时报价保存的深度。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# StopsLevel

获取订单距现价的最小距离（以点数为单位）。

```
int  StopsLevel() const
```

返回值

订单距现价的最小距离（以点数为单位）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# FreezeLevel

获取冻结位（以点数为单位）。

```
int  FreezeLevel() const
```

返回值

冻结位的距离（以点数为单位）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Bid

获取当前卖出价格。

```
double  Bid() const
```

返回值

当前卖出价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# BidHigh

获取日内最高卖出价格。

```
double  BidHigh() const
```

返回值

日内最高卖出价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# BidLow

获取日内最低卖出价格。

```
double  BidLow() const
```

返回值

日内最低卖出价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Ask

获取当前买入价格。

```
double  Ask() const
```

返回值

当前买入价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# AskHigh

获取日内最高买入价格。

```
double  AskHigh() const
```

返回值

日内最高买入价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# AskLow

获取日内最低买入价格。

```
double  AskLow() const
```

返回值

日内最低买入价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Last

获取当前最后价格。

```
double  Last() const
```

返回值

当前最后价格。

© 2000-2025, MetaQuotes Ltd.

# LastHigh

获取日内最高最后价格。

```
double  LastHigh() const
```

返回值

日内最高最后价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# LastLow

获取日内最低最后价格。

```
double  LastLow() const
```

返回值

日内最低最后价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeCalcMode

获取合约成本计算模式。

```
ENUM_SYMBOL_CALC_MODE  TradeCalcMode() const
```

返回值

合约成本计算模式。(值为 ENUM_SYMBOL_CALC_MODE 枚举)。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeCalcModeDescription

获取合约成本计算模式的字符串描述。

```
string  TradeCalcModeDescription() const
```

返回值

合约成本计算模式的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeMode

获取订单执行类型。

```
ENUM_SYMBOL_TRADE_MODE  TradeMode() const
```

返回值

订单执行类型（值为 ENUM_SYMBOL_TRADE_MODE 枚举）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeModeDescription

获取交易模式的字符串描述。

```
string  TradeModeDescription() const
```

返回值

交易模式的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeExecution

获取交易执行模式。

```
ENUM_SYMBOL_TRADE_EXECUTION  TradeExecution() const
```

返回值

交易执行模式（值为 ENUM_SYMBOL_TRADE_EXECUTION 枚举）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeExecutionDescription

获取交易执行模式的字符串描述。

```
string  TradeExecutionDescription() const
```

返回值

交易执行模式的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapMode

获取掉期利率的计算模式。

```
ENUM_SYMBOL_SWAP_MODE  SwapMode() const
```

返回值

掉期利率的计算模式（值为 ENUM_SYMBOL_SWAP_MODE 枚举）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapModeDescription

获取掉期利率的计算模式的字符串描述。

```
string  SwapModeDescription() const
```

返回值

掉期利率的计算模式的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapRollover3days

获取掉期利率滚动日。

```
ENUM_DAY_OF_WEEK  SwapRollover3days() const
```

返回值

掉期利率滚动日（值为 ENUM_DAY_OF_WEEK 枚举）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapRollover3daysDescription

获取掉期利率滚动日的字符串描述。

```
string  SwapRollover3daysDescription() const
```

返回值

掉期利率滚动日的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginInitial

获取初始保证金值。

```
double  MarginInitial()
```

返回值

初始保证金值。

注释

一手交易的保证金额度（以金融工具的货币保证金为单位）。当客户入场时，用于检查客户的净值。

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginMaintenance

获取维护保证金值。

```
double  MarginMaintenance()
```

返回值

维护保证金值。

注释

一手交易的保证金额度（以金融工具的货币保证金为单位）。当账户状态改变时，用于检查客户的净值。如果维护保证金等于 0，则使用初始保证金。

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginLong

获取支付多头仓位的保证金率。

```
double  MarginLong() const
```

返回值

获取支付多头仓位的保证金率。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginShort

获取支付控头仓位的保证金率。

```
double  MarginShort() const
```

返回值

支付空头仓位的保证金率。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginLimit

获取支付限价单的保证金率。

```
double  MarginLimit() const
```

返回值

支付限价单的保证金率。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginStop

获取支付突破单的保证金率。

```
double  MarginStop() const
```

返回值

支付突破单的保证金率。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# MarginStopLimit

获取支付突破/限价单的保证金率。

```
double  MarginStopLimit() const
```

返回值

支付突破/限价单的保证金率。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeTimeFlags

获取允许的订单过期模式标志。

```
int  TradeTimeFlags() const
```

返回值

允许的订单过期模式标志。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TradeFillFlags

获取允许的订单填充模式标志。

```
int  TradeFillFlags() const
```

返回值

允许的订单填充模式标志。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Digits

小数点之后的位数。

```
int  Digits() const
```

返回值

小数点之后的位数。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Point

获取点数值。

```
double  Point() const
```

返回值

点数值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TickValue

获取即时报价成本（最小价格变化）。

```
double  TickValue() const
```

返回值

即时报价成本（最小价格变化）。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TickValueProfit

获取经计算的盈利仓位的即时价格。

```
double  TickValueProfit() const
```

返回值

经计算的盈利仓位的即时价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TickValueLoss

获取经计算的亏损仓位的即时价格。

```
double  TickValueLoss() const
```

返回值

经计算的亏损仓位的即时价格。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TickSize

获取价格的最小变化。

```
double  TickSize() const
```

返回值

价格的最小变化。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# ContractSize

获取交易合约的额度。

```
double  ContractSize() const
```

返回值

交易合约的额度。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# LotsMin

获取一笔成交的最小成交量。

```
double  LotsMin() const
```

返回值

一笔成交的最小成交量

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# LotsMax

获取一笔成交的最大成交量。

```
double  LotsMax() const
```

返回值

一笔成交的最大成交量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# LotsStep

获取一笔成交的最小变化步长。

```
double  LotsStep() const
```

返回值

一笔成交的最小变化步长。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# LotsLimit

获取一个品种的最大允许开仓和挂单交易量 (方向无关)。

```
double  LotsLimit() const
```

返回值

一个品种的最大允许开仓和挂单交易量 (方向无关)。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapLong

获取多头持仓的掉期利率值。

```
double  SwapLong() const
```

返回值

多头持仓的掉期利率值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SwapShort

获取空头持仓的掉期利率值。

```
double  SwapShort() const
```

返回值

空头持仓的掉期利率值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# CurrencyBase

获取品名的基准货币。

```
string  CurrencyBase() const
```

返回值

品名的基准货币。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# CurrencyProfit

获取盈利货币名。

```
string  CurrencyProfit() const
```

返回值

盈利货币名。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# CurrencyMargin

获取保证金货币名。

```
string  CurrencyMargin() const
```

返回值

入金货币名。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Bank

获取当前报价来源的名称。

```
string  Bank() const
```

返回值

当前报价来源的名称。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# 描述

获取品种的字符串描述。

```
string  Description() const
```

返回值

品种的字符串描述。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Path

获取品种树的路径。

```
string  Path() const
```

返回值

获取品种树的路径。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionDeals

获取当前时段的成交单数量。

```
long  SessionDeals() const
```

返回值

当前时段的成交单数量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionBuyOrders

获取当前时刻的多头订单数量。

```
long  SessionBuyOrders() const
```

返回值

当前时刻的多头订单数量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionSellOrders

获取当前时刻的空头订单数量。

```
long  SessionSellOrders() const
```

返回值

当前时刻的空头订单数量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionTurnover

获取当前时段的流水摘要。

```
double  SessionTurnover() const
```

返回值

当前时段的流水摘要。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionInterest

获取当前时段的持仓摘要。

```
double  SessionInterest() const
```

返回值

当前时段的持仓摘要。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionBuyOrdersVolume

获取多头订单的当前交易量。

```
double  SessionBuyOrdersVolume() const
```

返回值

多头订单的当前交易量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionSellOrdersVolume

获取空头订单的当前交易量。

```
double  SessionSellOrdersVolume() const
```

返回值

空头订单的当前交易量。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionOpen

获取当前时段的开仓价。

```
double  SessionOpen() const
```

返回值

当前时段的开仓价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionClose

获取当前时段的平仓价。

```
double  SessionClose() const
```

返回值

当前时段的平仓价。<

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionAW

获取当前时段的平均加权价。

```
double  SessionAW() const
```

返回值

当前时段的平均加权价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionPriceSettlement

获取当前时段的结算价。

```
double  SessionPriceSettlement() const
```

返回值

当前时段的结算价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionPriceLimitMin

获取当前时段的最小价。

```
double  SessionPriceLimitMin() const
```

返回值

当前时段的最小价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# SessionPriceLimitMax

获取当前时段的最大价。

```
double  SessionPriceLimitMax() const
```

返回值

当前时段的最大价。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
bool  InfoInteger(
   ENUM_SYMBOL_INFO_INTEGER  prop_id,      // 属性 ID
   long&                     var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 整数型属性 ID (值为 ENUM_SYMBOL_INFO_INTEGER 枚举<t3>)。</t3>

*var*

[输出] 保存长整数型结果的变量的引用。

返回值

true -成功情况下，false -如果不能获取属性值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
bool  InfoDouble(
   ENUM_SYMBOL_INFO_DOUBLE  prop_id,      // 属性 ID
   double&                  var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 双精度型属性 ID (值为 ENUM_SYMBOL_INFO_DOUBLE 枚举)。

*var*

[输出] 保存双精度型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
bool  InfoString(
   ENUM_SYMBOL_INFO_STRING  prop_id,      // 属性 ID
   string&                  var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 文本属性 ID。

*var*

[输出] 保存字符串型结果的变量的引用。

返回值

true -成功情况下，false -如果不能获取属性值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# NormalizePrice

返回品种属性的规整化价格数值。

```
double  NormalizePrice(
   double  price   // 价格
   ) const
```

参数

*price*

[输入] 价位。

返回值

规整化价格数值。

注释

品种必须先行使用 Name 方法选择。

© 2000-2025, MetaQuotes Ltd.

# COrderInfo

COrderInfo 类用于简便地访问挂单属性。

## 描述

COrderInfo 类可供访问挂单属性。

## 声明

```
class COrderInfo : public CObject
```

## 标称库文件

```
#include <Trade\OrderInfo.mqh>
```

## 继承体系

CObject
  COrderInfo

## 类方法分组

| 访问整数型属性 | |
|---|---|
| Ticket | 获取之前已选择访问的订单单号 |
| TimeSetup | 获取订单放置时间 |
| TimeSetupMsc | 获取订单放置时间 数值是自1970 年 1 月 1 日以来的毫秒数 |
| OrderType | 获取订单类型 |
| OrderTypeDescription | 获取订单类型的字符串描述 |
| 状态 | 获取订单状态 |
| StateDescription | 获取订单状态的字符串描述 |
| TimeExpiration | 获取订单执行时间 |
| TimeDone | 获取订单执行或取消的时间 |
| TimeDoneMsc | 获取订单执行或取消的时间 数值是自1970 年 1 月 1 日以来的毫秒数 |
| TypeFilling | 以填充类型获取订单执行类型 |
| TypeFillingDescription | 以填充类型获取订单执行类型的字符串描述 |
| TypeTime | 获取过期订单的类型 |
| TypeTimeDescription | 获取过期订单的类型的字符串描述 |
| Magic | 获取放置订单的智能交易 ID |
| PositionId | 获取仓位 ID |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 访问整数型属性 | |
| 访问双精度型属性 | |
| VolumeInitial | 获取订单初始交易量 |
| VolumeCurrent | 获取订单的未填充交易量 |
| PriceOpen | 获取订单价格 |
| StopLoss | 获取订单的止损位 |
| TakeProfit | 获取订单的止盈位 |
| PriceCurrent | 以订单品种获取当前价格 |
| PriceStopLimit | 获取设置限价单的距离差价 |
| 访问文本型属性 | |
| Symbol | 获取订单品种的名称 |
| Comment | 获取订单注释 |
| 访问 MQL5 API函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值 |
| 状态 | |
| StoreState | 保存订单参数 |
| CheckState | 检查当前参数并与保存参数对比 |
| 选择 | |
| Select | 以单号选择订单，用于进一步访问其属性 |
| SelectByIndex | 以索引选择订单，用于进一步访问其属性 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Ticket

获取之前已通过 Select 方法选择访问的订单单号。

```
ulong  Ticket() const
```

返回值

如果成功，订单单号，否则 -ULONG_MAX。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeSetup

获取订单放置时间。

```
datetime  TimeSetup() const
```

返回值

订单放置时间。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeSetupMsc

获取订单放置执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

```
ulong  TimeSetupMsc() const
```

返回值

订单放置执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# OrderType

获取订单类型。

```
ENUM_ORDER_TYPE  OrderType()
```

返回值

订单类型（值为 ENUM_ORDER_TYPE 枚举）。

注释

订单必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeDescription

获取订单类型的字符串描述。

```
string  TypeDescription() const
```

返回值

订单类型的字符串描述。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# 状态

获取订单状态。

```
ENUM_ORDER_STATE  State() const
```

返回值

订单状态（值为 ENUM_ORDER_STATE 枚举）。

注释

订单必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# StateDescription

获取订单状态的字符串描述。

```
string  StateDescription() const
```

返回值

订单状态的字符串描述。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeExpiration

获取订单过期时间

```
datetime  TimeExpiration() const
```

返回值

放置订单时的过期时间。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeDone

获取订单执行或取消的时间。

```
datetime  TimeDone() const
```

返回值

订单执行或取消的时间。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeDoneMsc

获取订单执行或取消的时间，数值是自1970年1月1日以来的毫秒数。

```
ulong  TimeDoneMsc() const
```

返回值

订单执行或取消的时间，数值是自1970年1月1日以来的毫秒数。

注释

订单必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeFilling

获取订单填充类型。

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

返回值

订单填充类型（值为 ENUM_ORDER_TYPE_FILLING 枚举）。

注释

订单必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeFillingDescription

获取订单填充类型的字符串描述。

```
string  TypeFillingDescription() const
```

返回值

订单填充类型的字符串描述。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeTime

获取过期订单的类型。

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

返回值

过期订单的类型。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeTimeDescription

获取过期订单的类型的字符串描述。

```
string  TypeTimeDescription() const
```

返回值

过期订单的类型的字符串描述。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Magic

获取放置订单的智能交易 ID。

```
long  Magic() const
```

返回值

放置订单的智能交易 ID。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PositionId

获取仓位 ID。

```
long  PositionId() const
```

返回值

仓位对应 ID，即参与的订单。

注释

订单必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeInitial

获取订单初始交易量

```
double  VolumeInitial() const
```

返回值

订单初始交易量

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeCurrent

获取订单的未填充交易量。

```
double  VolumeCurrent() const
```

返回值

订单的未填充交易量。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceOpen

获取订单价格。

```
double  PriceOpen() const
```

返回值

订单放置价格。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# StopLoss

获取订单的止损位。

```
double  StopLoss() const
```

返回值

订单的止损位。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TakeProfit

获取订单的止盈位。

```
double  TakeProfit() const
```

返回值

Order's Take Profit.

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceCurrent

以订单品种获取当前价格。

```
double  PriceCurrent() const
```

返回值

订单品种的当前价格。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceStopLimit

获取设置限价单的距离差价。

```
double  PriceStopLimit() const
```

返回值

限价单的距离差价。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获取订单品种的名称。

```
string  Symbol() const
```

返回值

订单品种的名称。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Comment

获取订单注释。

```
string  Comment() const
```

返回值

订单注释。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,      // 属性 ID
   long&                        var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 整数型属性 ID (值为 ENUM_ORDER_PROPERTY_INTEGER 枚举)。

*var*

[输出] 保存长整数型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // 属性 ID
   double&                     var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 双精度型属性 ID (值为 ENUM_ORDER_PROPERTY_DOUBLE 枚举)。

*var*

[输出] 保存双精度型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,      // 属性 ID
   string&                     var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 文本属性 ID。

*var*

[输出] 保存字符串型结果的变量的引用。

返回值

true -成功情况下，false -如果不能获取属性值。

注释

订单必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# StoreState

保存订单参数。

```
void  StoreState()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CheckState

检查当前参数并与保存参数对比。

```
bool  CheckState()
```

返回值

true - 如果自上次调用 StoreState() 方法以来订单参数有变化，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Select

以单号选择订单，用于进一步访问其属性。

```
bool  Select(
   ulong  ticket   // 订单
   )
```

返回值

true -成功情况下 false -如果不能选择订单。

© 2000-2025, MetaQuotes Ltd.

# SelectByIndex

以索引选择订单，用于进一步访问其属性。

```
bool  SelectByIndex(
   int  index   // 订单索引
   )
```

返回值

true -成功情况下 false -如果不能选择订单。

© 2000-2025, MetaQuotes Ltd.

# CHistoryOrderInfo

CHistoryOrderInfo 类用于简便地访问历史订单属性。

## 描述

CHistoryOrderInfo 类可供访问历史订单属性。

## 声明

```
class CHistoryOrderInfo : public CObject
```

## 标称库文件

```
#include <Trade\HistoryOrderInfo.mqh>
```

## 继承体系

CObject

CHistoryOrderInfo

## 类方法分组

| 访问整数型属性 | |
|---|---|
| TimeSetup | 获取订单放置时间 |
| TimeSetupMsc | 获取订单放置时间 数值是自1970 年 1 月 1 日以来的毫秒数 |
| OrderType | 获取订单类型 |
| OrderTypeDescription | 获取订单类型的字符串描述 |
| 状态 | 获取订单状态 |
| StateDescription | 获取订单状态的字符串描述 |
| TimeExpiration | 获取订单执行时间 |
| TimeDone | 获取订单执行或取消的时间 |
| TimeDoneMsc | 获取订单执行或取消的时间 数值是自1970 年 1 月 1 日以来的毫秒数 |
| TypeFilling | 以填充类型获取订单执行类型 |
| TypeFillingDescription | 以填充类型获取订单执行类型的字符串描述 |
| TypeTime | 获取过期订单的类型 |
| TypeTimeDescription | 获取过期订单的类型的字符串描述 |
| Magic | 获取放置订单的智能交易 ID |
| PositionId | 获取仓位 ID |
| 访问双精度型属性 | |

© 2000-2025, MetaQuotes Ltd.

| 访问整数型属性 | |
|---|---|
| VolumeInitial | 获取订单初始交易量 |
| VolumeCurrent | 获取订单的未填充交易量 |
| PriceOpen | 获取订单价格 |
| StopLoss | 获取订单的止损位 |
| TakeProfit | 获取订单的止盈位 |
| PriceCurrent | 以订单品种获取当前价格 |
| PriceStopLimit | 获取设置限价单的距离差价 |
| 访问文本型属性 | |
| Symbol | 获取订单品名 |
| Comment | 获取订单注释 |
| 访问 MQL5 API函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值 |
| 选择 | |
| Ticket | 获取单号/选择订单 |
| SelectByIndex | 以索引选择订单 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# TimeSetup

获取订单放置时间。

```
datetime  TimeSetup() const
```

返回值

订单放置时间。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeSetupMsc

获取订单放置执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

```
ulong  TimeSetupMsc() const
```

返回值

订单放置执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# OrderType

获取订单类型。

```
ENUM_ORDER_TYPE  OrderType() const
```

返回值

订单类型（值为 ENUM_ORDER_TYPE 枚举）。

注释

历史订单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeDescription

获取订单类型的字符串描述。

```
string  TypeDescription() const
```

返回值

订单类型的字符串描述。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# 状态

获取订单状态。

```
ENUM_ORDER_STATE  State() const
```

返回值

订单状态（值为 ENUM_ORDER_STATE 枚举）。

注释

历史订单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# StateDescription

获取订单状态的字符串描述。

```
string  StateDescription() const
```

返回值

订单状态的字符串描述。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeExpiration

获取订单过期时间。

```
datetime  TimeExpiration() const
```

返回值

放置订单时的过期时间。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeDone

获取订单执行或取消的时间。

```
datetime  TimeDone() const
```

返回值

订单执行或取消的时间。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeDoneMsc

获取订单执行或取消的时间，数值是自1970年1月1日以来的毫秒数。

```
ulong  TimeDoneMsc() const
```

返回值

订单执行或取消的时间，数值是自1970年1月1日以来的毫秒数。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeFilling

以填充类型获取订单执行类型。

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

返回值

以填充类型获取订单执行类型（值为 ENUM_ORDER_TYPE_FILLING 枚举）。

注释

历史订单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeFillingDescription

以填充类型获取订单执行类型的字符串描述。

```
string  TypeFillingDescription() const
```

返回值

以填充类型获取订单执行类型的字符串描述。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeTime

获取过期订单的类型。

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

返回值

过期订单的类型（值为 ENUM_ORDER_TYPE_TIME 枚举）。

注释

历史订单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeTimeDescription

获取过期订单的类型的字符串描述。

```
string  TypeTimeDescription() const
```

返回值

过期订单的类型的字符串描述。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Magic

获取放置订单的智能交易 ID。

```
long  Magic() const
```

返回值

放置订单的智能交易 ID。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PositionId

获取仓位 ID。

```
long  PositionId() const
```

返回值

仓位对应 ID，即参与的订单。

注释

历史订单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeInitial

获取订单初始交易量

```
double  VolumeInitial() const
```

返回值

订单初始交易量

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# VolumeCurrent

获取订单的未填充交易量。

```
double  VolumeCurrent() const
```

返回值

订单的未填充交易量。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceOpen

获取订单价格。

```
double  PriceOpen() const
```

返回值

订单放置价格。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# StopLoss

获取订单的止损价。

```
double  StopLoss() const
```

返回值

订单的止损价。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TakeProfit

获取订单的止盈价。

```
double  TakeProfit() const
```

返回值

订单的止盈价。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceCurrent

获取订单对应品种的当前价格。

```
double  PriceCurrent() const
```

返回值

订单对应品种的当前价格。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceStopLimit

获取挂单距现价的距离。

```
double  PriceStopLimit() const
```

返回值

挂单距现价的距离。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获取订单品种的名称。

```
string  Symbol() const
```

返回值

订单品种的名称。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Comment

获取订单注释。

```
string  Comment() const
```

返回值

订单注释。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,      // 属性 ID
   long&                        var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 整数型属性 ID (值为 ENUM_ORDER_PROPERTY_INTEGER 枚举)。

*var*

[输出] 保存长整数型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // 属性 ID
   double&                     var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 双精度型属性 ID (值为 ENUM_ORDER_PROPERTY_DOUBLE 枚举)。

*var*

[输出] 保存双精度型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,      // 属性 ID
   string&                     var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 属性 ID (值为 ENUM_ORDER_PROPERTY_STRING 枚举)。

*var*

[输出] 保存字符串型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

历史订单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

## Ticket (Get方法)

获取订单单号。

```
ulong  Ticket() const
```

返回值

订单单号。

## Ticket (Set方法)

选择订单用于进一步操作。

```
void  Ticket(
   ulong  ticket      // 订单单号
   )
```

参数

*ticket*

[输入] 订单单号。

© 2000-2025, MetaQuotes Ltd.

# SelectByIndex

以索引选择订单，用于进一步访问其属性。

```
bool  SelectByIndex(
   int  index   // 订单索引
   )
```

返回值

true -成功情况下 false -如果不能选择订单。

© 2000-2025, MetaQuotes Ltd.

# CPositionInfo

CPositionInfo 类用于简便地访问已开仓位属性。

## 描述

CPositionInfo 类可供访问已开仓位属性。

## 声明

```
class CPositionInfo : public CObject
```

## 标称库文件

```
#include <Trade\PositionInfo.mqh>
```

## 继承体系

COBject

CPositionInfo

## 类方法分组

| 访问整数型属性 | |
|---|---|
| Time | 获取开仓时间 |
| TimeMsc | 获取开仓时间，数值是自1970 年 1 月 1 日以来的毫秒数 |
| TimeUpdate | 获取持仓变更时间，数值是自1970 年 1 月 1 日以来的秒数 |
| TimeUpdateMsc | 获取持仓变更时间，数值是自1970 年 1 月 1 日以来的秒数 |
| PositionType | 获取持仓类型 |
| TypeDescription | 获取持仓类型的字符串描述 |
| Magic | 获取开仓的智能交易 ID |
| 标识符 | 获取仓位 ID |
| 访问双精度型属性 | |
| Volume | 获取持仓成交量 |
| PriceOpen | 获取开仓价位 |
| StopLoss | 获取持仓的止损价 |
| TakeProfit | 获取持仓的止盈价 |
| PriceCurrent | 获取持仓品种当前价格 |
| Commission | 获取持仓的佣金额度 |
| Swap | 获取仓位的掉期利率额度 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 访问整数型属性 | |
| Profit | 获取持仓的当前盈利额度 |
| 访问文本型属性 | |
| Symbol | 获取持仓品种的名称 |
| Comment | 获取持仓的注释 |
| 访问 MQL5 API 函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值。 |
| 选择 | |
| Select | 选择持仓 |
| SelectByIndex | 根据索引选择持仓 |
| SelectByMagic | 根据指定交易品种名称和幻数选择持仓 |
| SelectByTicket | 根据价格选择持仓 |
| 状态 | |
| StoreState | 保存持仓参数 |
| CheckState | 检查当前参数并与保存参数对比 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Time

获取开仓时间。

```
datetime  Time() const
```

返回值

开仓时间。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeMsc

获取开仓的时间，数值是自1970 年 1 月 1 日以来的毫秒数。

```
ulong  TimeMsc() const
```

返回值

开仓的时间，数值是自1970 年 1 月 1 日以来的毫秒数。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeUpdate

获取持仓变更时间，数值是自1970年1月1日以来的秒数。

```
datetime  TimeUpdate() const
```

返回值

持仓变更时间，数值是自1970年1月1日以来的秒数。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeUpdateMsc

获取持仓变更时间，数值是自1970年1月1日以来的毫秒数。

```
ulong  TimeUpdateMsc() const
```

返回值

持仓变更时间，数值是自1970年1月1日以来的毫秒数。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# PositionType

获取持仓类型。

```
ENUM_POSITION_TYPE  PositionType() const
```

返回值

持仓类型（值为 ENUM_POSITION_TYPE 枚举）。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeDescription

获取持仓类型的字符串描述。

```
string  TypeDescription() const
```

返回值

持仓类型的字符串描述。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Magic

获取开仓的智能交易 ID。

```
long  Magic() const
```

返回值

执行开仓的智能交易 ID。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# 标识符

获取仓位 ID。

```
long  Identifier() const
```

返回值

持仓的ID。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Volume

获取持仓成交量。

```
double  Volume() const
```

返回值

持仓成交量。s

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceOpen

获取开仓价位。

```
double  PriceOpen() const
```

返回值

开仓价。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# StopLoss

获取持仓的止损价。

```
double  StopLoss() const
```

返回值

持仓的止损价。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TakeProfit

获取持仓的止盈价。

```
double  TakeProfit() const
```

返回值

持仓的止盈价。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PriceCurrent

获取持仓品种当前价格。

```
double  PriceCurrent() const
```

返回值

持仓品种的当前价格。

© 2000-2025, MetaQuotes Ltd.

# Commission

获取持仓的佣金额度。

```
double  Commission() const
```

返回值

持仓的佣金额度（以入金货币为单位）。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# Swap

获取持仓的掉期利率额度。

```
double  Swap() const
```

返回值

持仓的掉期利率额度（以入金货币为单位）。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# Profit

获取持仓的当前盈利额度。

```
double  Profit() const
```

返回值

持仓的当前盈利额度（以入金货币为单位）。

注释

持仓必须先行使用 Select（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获取持仓品种的名称。

```
string  Symbol() const
```

返回值

持仓品种的名称。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Comment

获取持仓的注释。

```
string  Comment() const
```

返回值

持仓的注释。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
bool  InfoInteger(
   ENUM_POSITION_PROPERTY_INTEGER  prop_id,      // 属性 ID
   long&                           var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 整数型属性 ID (值为 ENUM_POSITION_PROPERTY_INTEGER 枚举)。

*var*

[输出] 保存长整数型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
bool  InfoDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  prop_id,      // 属性 ID
   double&                        var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 双精度型属性 ID (值为 ENUM_POSITION_PROPERTY_DOUBLE 枚举)。

*var*

[输入] 保存双精度型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
bool  InfoString(
   ENUM_POSITION_PROPERTY_STRING  prop_id,      // 属性 ID
   string&                        var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 文本属性 ID (值为 ENUM_POSITION_PROPERTY_STRING 枚举)。

*var*

[输出] 保存字符串型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

持仓必须先行使用 Select (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Select

选择仓位用于进一步操作。

```
bool  Select(
   const string  symbol      // 品名
   )
```

参数

*symbol*

[输入] 选择持仓的品名。

© 2000-2025, MetaQuotes Ltd.

# SelectByIndex

以索引选择持仓，用于进一步访问属性。

```
bool  SelectByIndex(
   int  index   // 持仓索引
   );
```

返回值

true -成功情况下 false -如果不能选择持仓。

© 2000-2025, MetaQuotes Ltd.

# SelectByMagic

根据金融工具名称和未来工作的幻数选择持仓。

```
bool  SelectByMagic(
   const string  symbol, // 交易品种名称
   const ulong   magic   // 幻数
   );
```

参数

*交易品种*

[in] 交易品种名称。

*幻数*

[in] 持仓的幻数。

返回值

如果成功返回true，如果选择持仓失败返回。

© 2000-2025, MetaQuotes Ltd.

# SelectByTicket

通过报价选择持仓用于进一步操作。

```
bool  SelectByTicket(
   ulong  ticket      // 持仓报价
   );
```

参数

*报价*

[in] 持仓报价。

返回值

true - 成功，false - 未选择持仓。

© 2000-2025, MetaQuotes Ltd.

# StoreState

保存持仓参数。

```
void  StoreState()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CheckState

检查当前参数并与保存参数对比。

```
bool  CheckState()
```

返回值

true - 如果自上次调用 StoreState() 方法以来仓位参数有变化，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# CDealInfo

CDealInfo 类用于简便地访问成交单属性。

## 描述

CDealInfo 类可供访问成交单属性。

## 声明

```
class CDealInfo : public CObject
```

## 标称库文件

```
#include <Trade\DealInfo.mqh>
```

## 继承体系

CObject
  CDealInfo

## 类方法分组

| 访问整数型属性 | |
|---|---|
| Order | 获取已执行并成交的订单 |
| Time | 获取成交单时间 |
| TimeMsc | 获取 成交单执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。 |
| DealType | 获取成交单类型 |
| TypeDescription | 获取成交单类型的字符串描述 |
| Entry | 获取成交单方向 |
| EntryDescription | 获取成交单方向的字符串描述 |
| Magic | 获取执行成交单的智能交易 ID |
| PositionId | 获取仓位对应 ID，即参与的成交单 |
| 访问双精度型属性 | |
| Volume | 获取成交单成交量 |
| Price | 获取成交单价格 |
| Commision | 获取成交单的佣金额度 |
| Swap | 获取仓位平仓时的掉期利率额度 |
| Profit | 获取成交单的财会结果 |
| 访问文本型属性 | |

© 2000-2025, MetaQuotes Ltd.

| 访问整数型属性 | |
|---|---|
| Symbol | 获取成交单品种的名称 |
| Comment | 获取成交单注释 |
| 访问 MQL5 API函数 | |
| InfoInteger | 获取指定整数型属性的值 |
| InfoDouble | 获取指定双精度型属性的值 |
| InfoString | 获取指定字符串型属性的值 |
| 选择 | |
| Ticket | 获取单号/选择成交单 |
| SelectByIndex | 以索引选择成交单 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Order

获取已执行并成交的订单。

```
long  Order() const
```

返回值

已执行并成交的订单。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Time

获取成交单时间。

```
datetime  Time() const
```

返回值

成交单时间。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# TimeMsc

获取成交单执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

```
ulong  TimeMsc() const
```

返回值

成交单执行时间，数值是自1970 年 1 月 1 日以来的毫秒数。

注释

成交单必须先行使用 Ticket (通过单号）或 SelectByIndex (通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# DealType

获取成交单类型。

```
ENUM_DEAL_TYPE  DealType() const
```

返回值

成交单类型（值为 ENUM_DEAL_TYPE 枚举）。

注释

成交单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# TypeDescription

获取成交单类型的字符串描述。

```
string  TypeDescription() const
```

返回值

成交单类型的字符串描述。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Entry

获取成交单方向。

```
ENUM_DEAL_ENTRY  Entry() const
```

返回值

成交单方向（值为 ENUM_DEAL_ENTRY 枚举）。

注释

成交单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# EntryDescription

获取成交单方向的字符串描述。

```
string  EntryDescription() const
```

返回值

成交单方向的字符串描述。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Magic

获取执行成交单的智能交易 ID。

```
long  Magic() const
```

返回值

执行成交单的智能交易 ID。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# PositionId

获取仓位对应 ID，即参与的成交单。

```
long  PositionId() const
```

返回值

仓位对应 ID，即参与的成交单。

注释

成交单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# Volume

获取成交单成交量。

```
double  Volume() const
```

返回值

成交单成交量。<s

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Price

获取成交单价格。

```
double  Price() const
```

返回值

成交单价格。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Commission

获取成交单的佣金额度。

```
double  Commission() const
```

返回值

成交单的佣金额度。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Swap

获取仓位平仓时的掉期利率额度。

```
double  Swap() const
```

返回值

仓位平仓时的掉期利率额度。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Profit

获取成交单的财会结果。

```
double  Profit() const
```

返回值

成交单的财会结果（以入金货币为单位）。

注释

成交单必须先行使用 Ticket（通过单号）或 SelectByIndex（通过索引）方法选择。

© 2000-2025, MetaQuotes Ltd.

# Symbol

获取成交单品种的名称。

```
string  Symbol() const
```

返回值

成交单品种的名称。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# Comment

获取成交单注释。

```
string  Comment() const
```

返回值

成交注释。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

获取指定整数型属性的值。

```
bool  InfoInteger(
   ENUM_DEAL_PROPERTY_INTEGER  prop_id,      // 属性 ID
   long&                       var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 整数型属性 ID (值为 ENUM_DEAL_PROPERTY_INTEGER 枚举)。

*var*

[输出] 保存长整数型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoDouble

获取指定双精度型属性的值。

```
bool  InfoDouble(
   ENUM_DEAL_PROPERTY_DOUBLE  prop_id,      // 属性 ID
   double&                    var           // 变量引用
   ) const
```

参数

*prop_id*

[输入] 双精度型属性 ID (值为 ENUM_DEAL_PROPERTY_DOUBLE 枚举)。

*var*

[输出] 保存双精度型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

# InfoString

获取指定字符串型属性的值。

```
bool  InfoString(
   ENUM_DEAL_PROPERTY_STRING  prop_id,      // 属性 ID
   string&                    var           // 变量引用
   ) const
```

参数

*prop_id*

[输入]属性 ID (值为 ENUM_DEAL_PROPERTY_STRING 枚举)。

*var*

[输出] 保存字符串型结果的变量的引用。

返回值

true -成功情况下, false -如果不能获取属性值。

注释

成交单必须先行使用 Ticket (通过单号) 或 SelectByIndex (通过索引) 方法选择。

© 2000-2025, MetaQuotes Ltd.

## Ticket (Get方法)

获取成交单单号。

```
ulong  Ticket() const
```

返回值

成交单单号。

## Ticket (Set方法)

选择仓位用于进一步操作。

```
void  Ticket(
   ulong  ticket      // 单号
   )
```

参数

*ticket*

[输入] 成交单单号。

© 2000-2025, MetaQuotes Ltd.

# SelectByIndex

以索引选择成交单，用于进一步访问属性。

```
bool  SelectByIndex(
   int  index   // 订单索引
   )
```

返回值

true -成功情况下 false -如果不能选择成交单。

© 2000-2025, MetaQuotes Ltd.

# CTrade

CTrade 类用于简便地访问交易函数。

## 描述

CTrade 类可供简便地访问交易函数。

## 声明

```
class CTrade : public CObject
```

## 标称库文件

```
#include <Trade\Trade.mqh>
```

## 继承体系

CObject

CTrade

直接分支

CExpertTrade

## 类方法分组

| 设置参数 | |
|---|---|
| LogLevel | 设置日志级别 |
| SetExpertMagicNumber | 设置智能 ID |
| SetDeviationInPoints | 设置允许的滑点 |
| SetTypeFilling | 设置订单填充类型 |
| SetTypeFillingBySymbol | 根据指定交易品种的设置来设定订单的成交指令类型 |
| SetAsyncMode | 设置交易操作的异步模式 |
| SetMarginMode | 根据当前账户设置来设定预付款计算模式 |
| 操作订单 | |
| OrderOpen | 按照设置参数放置订单 |
| OrderModify | 修改挂单参数 |
| OrderDelete | 删除挂单 |
| 操作仓位 | |
| PositionOpen | 按照设置参数开仓 |
| PositionModify | 修改仓位参数 |

© 2000-2025, MetaQuotes Ltd.

| 设置参数 | |
|---|---|
| PositionClose | 平仓 |
| PositionClosePartial | 部分平仓指定交易品种或指定单号 |
| PositionCloseBy | 根据反向持仓关闭指定价格的持仓 |
| 附加方法 | |
| Buy | 按照指定参数开多头仓位 |
| Sell | 按照指定参数开空头仓位 |
| BuyLimit | 按照指定参数放置低多类型挂单 |
| BuyStop | 按照指定参数放置高多类型挂单 |
| SellLimit | 按照指定参数放置高空类型挂单 |
| SellStop | 按照指定参数放置低空类型挂单 |
| 访问最后的请求参数 | |
| Request | 获取最后请求结构的拷贝 |
| RequestAction | 获取交易操作类型 |
| RequestActionDescription | 获取交易操作类型的字符串描述 |
| RequestMagic | 获取智能交易的幻数 |
| RequestOrder | 获取最后一笔请求使用的订单号 |
| RequestSymbol | 获取最后一笔请求使用的品种名称 |
| RequestVolume | 获取最后一笔请求使用的交易量（手数为单位） |
| RequestPrice | 获取最后一笔请求使用的价格 |
| RequestStopLimit | 获取最后一笔请求使用的高空挂单的价位 |
| RequestSL | 获取最后一笔请求使用的订单止损价 |
| RequestTP | 获取最后一笔请求使用的订单止盈价 |
| RequestDeviation | 获取最后一笔请求使用的订单价格滑点 |
| RequestType | 获取最后一笔请求使用的订单类型 |
| RequestTypeDescription | 获取最后一笔请求使用的订单类型的字符串描述 |
| RequestTypeFilling | 获取最后一笔请求使用的订单填充类型 |
| RequestTypeFillingDescription | 获取最后一笔请求使用的订单填充类型的字符串描述 |
| RequestTypeTime | 获取最后一笔请求使用的订单有效期 |
| RequestTypeTimeDescription | 获取最后一笔请求使用的订单有效期的字符串描述 |
| RequestExpiration | 获取最后一笔请求使用的订单过期时间 |

© 2000-2025, MetaQuotes Ltd.

| 设置参数 | |
|---|---|
| RequestComment | 获取最后一笔请求使用的订单注释 |
| RequestPosition | 获得持仓价格 |
| RequestPositionBy | 获得反向持仓价格 |
| 访问最后请求的检查结果 | |
| CheckResult | 获取最后请求检查结果的结构拷贝 |
| CheckResultRetcode | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段的值 |
| CheckResultRetcodeDescription | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段值的字符串描述 |
| CheckResultBalance | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 balance 字段的值 |
| CheckResultEquity | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 equity 字段的值 |
| CheckResultProfit | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 profit 字段的值 |
| CheckResultMargin | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin 字段的值 |
| CheckResultMarginFree | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_free 字段的值 |
| CheckResultMarginLevel | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_level字段的值 |
| CheckResultComment | 获取检查请求正确性时填充的 MqlTradeCheckResult 类型 comment 字段的值 |
| 访问最后请求的执行结果 | |
| Result | 获取最后请求结果的结构拷贝 |
| ResultRetcode | 获取请求结果的代码 |
| ResultRetcodeDescription | 获取请求结果代码的文本 |
| ResultDeal | 获取成交单单号。 |
| ResultOrder | 获取订单单号。 |
| ResultVolume | 获取成交单或订单的交易量 |
| ResultPrice | 获取券商确认的价格 |
| ResultBid | 获取当前卖出价格（报价） |
| ResultAsk | 获取当前买入价格（报价） |
| ResultComment | 获取券商注释 |

© 2000-2025, MetaQuotes Ltd.

| 设置参数 | |
|---|---|
| 辅助方法 | |
| PrintRequest | 在日志里输出最后一笔请求的参数 |
| PrintResult | 在日志里输出最后一笔请求的结果 |
| FormatRequest | 准备最后一笔请求参数的格式字符串 |
| FormatRequestResult | 准备最后一笔请求执行结果的格式字符串 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# LogLevel

设置消息日志级别。

```
void  LogLevel(
   ENUM_LOG_LEVELS  log_level      // 日志级别
   )
```

参数

*log_level*

[输入] 日志级别。

返回值

无。

注释

LOG_LEVEL_NO 及以下禁止显示任何消息 (在优化模式里自动设置)。LOG_LEVEL_ERRORS 仅允许显示出错消息 (省缺)。LOG_LEVEL_ALL 及以上允许显示所有消息 (在测试模式里自动设置)。

**ENUM_LOG_LEVELS**

| 标识符 | 描述 | 数值 |
|---|---|---|
| LOG_LEVEL_NO | 禁止显示消息 | 0 |
| LOG_LEVEL_ERRORS | 仅显示出错消息 | 1 |
| LOG_LEVEL_ALL | 显示所有消息 | 2 |

© 2000-2025, MetaQuotes Ltd.

# SetExpertMagicNumber

设置智能交易 ID。

```
void  SetExpertMagicNumber(
   ulong  magic      // ID
   )
```

参数

*magic*

[输入] 智能交易的新 ID。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# SetDeviationInPoints

设置允许的滑点。

```
void  SetDeviationInPoints(
   ulong  deviation      // 滑点
   )
```

参数

*deviation*

[输入] 允许的滑点。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# SetTypeFilling

设置订单填充类型。

```
void  SetTypeFilling(
   ENUM_ORDER_TYPE_FILLING  filling      // 订单填充类型
   )
```

参数

*filling*

[输入] 订单填充类型（值为 ENUM_ORDER_TYPE_FILLING 枚举）。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# SetTypeFillingBySymbol

根据指定交易品种的设置来设定订单的成交指令类型。

```
bool  SetTypeFillingBySymbol(
   const string   symbol      // 交易品种名称
   )
```

参数

*交易品种*

[in] 交易品种名称，SYMBOL_FILLING_MODE包括可允许的订单成交指令。

返回值

true -成功执行，false -定义成交指令失败。

注意

如果一个交易品种同时允许SYMBOL_FILLING_FOK和SYMBOL_FILLING_IOC成交指令，则为订单设置ORDER_FILLING_FOK值。

© 2000-2025, MetaQuotes Ltd.

# SetAsyncMode

设置交易操作的异步模式。

```
void  SetAsyncMode(
   bool  mode      // 异步模式标志
  )
```

参数

*mode*

[输入] 异步模式标志。

返回值

无。

注释

此模式用于异步交易操作 (无需等待交易服务器响应发送请求) (参阅 OrderSendAsync)。

© 2000-2025, MetaQuotes Ltd.

# SetMarginMode

根据当前账户设置来设定预付款计算模式。

```
void  SetMarginMode()
```

返回值

No.

注意

在ENUM_ACCOUNT_MARGIN_MODE指定的预付款计算模式。

© 2000-2025, MetaQuotes Ltd.

# OrderOpen

按照设置参数放置订单。

```
bool  OrderOpen(
   const string           symbol,           // 品名
   ENUM_ORDER_TYPE        order_type,       // 订单类型
   double                 volume,           // 订单交易量
   double                 limit_price,      // StopLimit 价格
   double                 price,            // 执行价格
   double                 sl,               // 止损价
   double                 tp,               // 止盈价
   ENUM_ORDER_TYPE_TIME   type_time,        // 过期类型
   datetime               expiration,       // 过期tion
   const string           comment=""        // 注释
   )
```

参数

*symbol*

[输入] 交易工具名称。

*order_type*

[输入] 交易操作类型 (值为 ENUM_ORDER_TYPE 枚举)。

*volume*

[输入] 请求的订单交易量。

*limit_price*

[输入] 放置 StopLimit 挂单的价位。

*price*

[输入] 订单必须被执行的价格。

*sl*

[输入] 触发止损的价格。

*tp*

[输入] 触发止盈的价格。

*type_time*

[输入] 订单执行类型 (值为 ENUM_ORDER_TYPE_TIME 枚举)。

*expiration*

[输入] 挂单执行日期。

*comment=""*

[输入] 订单注释。

返回值

true -结构检查基本成功情况下，否则 false。

© 2000-2025, MetaQuotes Ltd.

注释

成功完成 OrderSend(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器返回代码），即 ResultRetcode() 和 <li5>ResultDeal()</li5> 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# OrderModify

修改挂单参数。

```
bool  OrderModify(
   ulong                 ticket,          // 订单单号
   double                price,           // 执行价格
   double                sl,              // 止损价
   double                tp,              // 止盈价
   ENUM_ORDER_TYPE_TIME  type_time,       // 过期类型
   datetime              expiration,      // 过期
   double                stoplimit        // 限价单价格
   )
```

参数

*ticket*

[输入] 订单单号。

*price*

[输入] 订单必须执行的新价格 (或之前值，如果没必要改变)。

*sl*

[输入] 触发止损的新价格 (或之前值，如果没必要改变)。

*tp*

[输入] 触发止盈的新价格 (或之前值，如果没必要改变)。

*type_time*

[输入] 订单过期的新类型 (或之前值，如果没必要改变)，值为 ENUM_ORDER_TYPE_TIME 枚举。

*expiration*

[输入] 挂单的新过期时间 (或之前值，如果没必要改变)。

*stoplimit*

[输入] 用于设置限价挂单的新价格，当价格抵达 *price* 值的时候。它仅用于指定 StopLimit 单。

返回值

true -结构检查基本成功情况下，否则 false。

注释

成功完成 OrderModify(...) 法并不意味着总是成功执行交易操作。有必要检查交易请求的结果 (交易服务器返回码)，使用 ResultRetcode() 获取。

© 2000-2025, MetaQuotes Ltd.

# OrderDelete

删除挂单。

```
bool  OrderDelete(
   ulong  ticket      // 订单单号
   )
```

参数

*ticket*

[输入] 订单单号。

返回值

true -结构检查基本成功情况下，否则 false。

注释

成功完成 OrderDelete(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果 (交易服务器返回码)，使用 ResultRetcode() 获取。

© 2000-2025, MetaQuotes Ltd.

# PositionOpen

按照设置参数开仓。

```
bool  PositionOpen(
   const string      symbol,          // 品名
   ENUM_ORDER_TYPE   order_type,      // 开仓的订单类型
   double            volume,          // 开仓交易量
   double            price,           // 执行价格
   double            sl,              // 止损价
   double            tp,              // 止盈价
   const string      comment=""       // 注释
   )
```

参数

*symbol*

[输入] 交易工具名称，即用来开仓的。

*order_type*

[输入] 开仓的订单类型（交易操作）(值为 ENUM_ORDER_TYPE 枚举)。

*volume*

[输入]请求的开仓交易量。

*price*

[输入]开仓的价位。

*sl*

[输入]触发止损的价格。

*tp*

[输入]触发止盈的价格。

*comment=""*

[输入] 仓位注释。

返回值

true -结构检查基本成功情况下，否则 false。

注释

成功完成 PositionOpen(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器返回代码），即 ResultRetcode() 和 ResultDeal()返回的代码。

© 2000-2025, MetaQuotes Ltd.

# PositionModify

以指定交易品种名称修改仓位参数。

```
bool  PositionModify(
   const string  symbol,      // 交易品种
   double        sl,          // 止损价
   double        tp           // 止盈价
   )
```

根据指定价格更改持仓参数。

```
bool  PositionModify(
   const ulong   ticket,      // 持仓报价
   double        sl,          // 止损价
   double        tp           // 止赢价
   )
```

参数

*交易品种*

[输入] 交易工具名称，即用来修改的。

*报价*

[in] 持仓报价将被更改。

*sl*

[输入] 触发止损的新价格（或之前值，如果没必要改变）。

*tp*

[输入] 触发止盈的新价格（或之前值，如果没必要改变）。

返回值

true -结构检查基本成功情况下，否则 false。

注释

成功完成 PositionModify(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器返回码），使用 ResultRetcode() 获取。

© 2000-2025, MetaQuotes Ltd.

# PositionClose

以指定交易品种平仓。

```
bool  PositionClose(
   const string  symbol,                  // 交易品种
   ulong         deviation=ULONG_MAX      // 滑点
   )
```

关闭指定报价的持仓。

```
bool  PositionClose(
   const ulong   ticket,                  // 持仓价格
   ulong         deviation=ULONG_MAX      // 偏差
   )
```

参数

*交易品种*

[输入] 交易工具名称，即用来平仓的。

*报价*

[in] 平仓价格。

*deviation=ULONG_MAX*

[输入] 距现价的最大滑点（点数为单位）。

返回值

true -结构检查基本成功情况下，否则 false。

注释

成功完成 PositionClose(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器返回码），使用 ResultRetcode() 获取。

© 2000-2025, MetaQuotes Ltd.

# PositionClosePartial

"锁仓"账户情况下，部分平仓指定交易品种。

```
bool  PositionClosePartial(
   const string  symbol,                  // 交易品种
   const double  volume,                  // 交易量
   ulong         deviation=ULONG_MAX      // 偏差
   )
```

"锁仓"账户情况下，部分平仓指定单号。

```
bool  PositionClosePartial(
   const ulong   ticket,                  // 持仓单号
   const double  volume,                  // 交易量
   ulong         deviation=ULONG_MAX      // 偏差
   )
```

参数

*交易品种*

[in] 部分平仓的交易工具名称。如果一个交易品种（不是单号）被指定部分平仓，那么会选择该交易品种第一个检测到的具有指定幻数（EA交易ID）的持仓。因此，有时通过指定持仓单号使用PositionClosePartial()会更好。

*交易量*

[in] 交易量，应据此减少持仓。如果该值超过部分平仓的交易量，则全部平仓。没有反向持仓。

*单号*

[in] 平仓单号。

*偏差=ULONG_MAX*

[in] 当前价格的最大偏差（以点数为单位）。

返回值

如果基本检测架构成功，返回true，否则返回false。

注意

成功完成PositionClosePartial(...)方法并非始终意味着成功执行交易操作。您应该调用ResultRetcode()方法来检测交易请求的结果（交易服务器返回代码）。

在"单边"系统中（ACCOUNT_MARGIN_MODE_RETAIL_NETTING和ACCOUNT_MARGIN_MODE_EXCHANGE），对于每个交易品种，在任何给定的时刻，只能打开一个成为一笔或多笔交易结果的持仓。不要混淆也显示在客户端"工具箱"面板"交易"标签上的当前持仓挂单。

在持仓表示下（ACCOUNT_MARGIN_MODE_RETAIL_HEDGING），每个交易品种可能同时打开多个持仓。在这种情况下，PositionClose关闭最小单号持仓。

© 2000-2025, MetaQuotes Ltd.

# PositionCloseBy

根据反向持仓关闭指定价格的持仓

```
bool  PositionCloseBy(
   const ulong   ticket,         // 持仓价格
   const ulong   ticket_by       // 反向持仓价格
   )
```

参数

*报价*

[in] 平仓价格。

*ticket_by*

[in] 反向持仓的关闭价格。

返回值

如果结构的基本检查成功，返回true，否则是false。

注意

成功完成PositionCloseBy(...) 类函数并不始终表示成功执行交易操作。您可以调用 ResultRetcode() 类函数，检查交易请求结果（交易服务器返回代码）。

© 2000-2025, MetaQuotes Ltd.

# Buy

按照指定参数开多头仓位。

```
bool  Buy(
   double         volume,           // 开仓成交量
   const string   symbol=NULL,      // 品名
   double         price=0.0,        // 价格
   double         sl=0.0,           // 止损位
   double         tp=0.0,           // 止盈价
   const string   comment=""        // 注释
   )
```

参数

*volume*

[输入] 开仓成交量。

*symbol=NULL*

[输入] 开仓品名。如果品名未指定，则使用当前品名。

*price=0.0*

[输入] 价位。如果价格未指定，则使用当前市场的买入价格。

*sl=0.0*

[输入] 止损价位。

*tp=0.0*

[输入] 止盈价位。

*comment=""*

[输入] 注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 Buy(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultDeal() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# Sell

按照指定参数开空头仓位。

```
bool  Sell(
   double         volume,           // 开仓成交量
   const string   symbol=NULL,      // 品名
   double         price=0.0,        // 价格
   double         sl=0.0,           // 止损位
   double         tp=0.0,           // 止盈价
   const string   comment=""        // 注释
   )
```

参数

*volume*

[输入] 开仓成交量。

*symbol=NULL*

[输入] 开仓品名。如果品名未指定，则使用当前品名。

*price=0.0*

[输入] 价位。如果价格未指定，则使用当前市场的卖出价格。

*sl=0.0*

[输入] 止损价位。

*tp=0.0*

[输入] 止盈价位。

*comment=""*

[输入] 注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 Sell(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultDeal() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# BuyLimit

按照指定参数放置低多类型挂单（在低于市场现价的价位买入）。

```
bool  BuyLimit(
   double                volume,                    // 订单交易量
   double                price,                     // 订单价格
   const string          symbol=NULL,               // 品名
   double                sl=0.0,                    // 止损价
   double                tp=0.0,                    // 止盈价
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,  // 订单生命周期
   datetime              expiration=0,              // 订单过期时间
   const string          comment=""                 // 注释
   )
```

参数

*volume*

[输入] 订单交易量。

*price*

[输入] 订单价格。

*symbol=NULL*

[输入] 订单品名。如果品名未指定，则使用当前品名。

*sl=0.0*

[输入] 止损价位。

*tp=0.0*

[输入] 止盈价位。

*type_time=ORDER_TIME_GTC*

[输入] 订单生命周期（值为 ENUM_ORDER_TYPE_TIME 枚举）。

*expiration=0*

[输入] 订单过期时间（仅用在 type_time=ORDER_TIME_SPECIFIED）。

*comment=""*

[输入] 订单注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 BuyLimit(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultOrder() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# BuyStop

按照指定参数放置高多类型挂单（在高于市场现价的价位买入）。

```
bool  BuyStop(
   double                volume,                    // 订单交易量
   double                price,                     // 订单价格
   const string          symbol=NULL,               // 品名
   double                sl=0.0,                    // 止损价
   double                tp=0.0,                    // 止盈价
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,  // 订单生命周期
   datetime              expiration=0,              // 订单过期时间
   const string          comment=""                 // 注释
   )
```

参数

*volume*
[输入] 订单交易量。

*price*
[输入] 订单价格。

*symbol=NULL*
[输入] 订单品名。如果品名未指定，则使用当前品名。

*sl=0.0*
[输入] 止损价位。

*tp=0.0*
[输入] 止盈价位。

*type_time=ORDER_TIME_GTC*
[输入] 订单生命周期（值为 ENUM_ORDER_TYPE_TIME 枚举）。

*expiration=0*
[输入] 订单过期时间（仅用在 type_time=ORDER_TIME_SPECIFIED）。

*comment=""*
[输入] 订单注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 BuyStop(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultOrder() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# SellLimit

按照指定参数放置高空类型挂单（在高于市场现价的价位卖出）。

```
bool  SellLimit(
   double                volume,                    // 订单交易量
   double                price,                     // 订单价格
   const string          symbol=NULL,               // 品名
   double                sl=0.0,                    // 止损价
   double                tp=0.0,                    // 止盈价
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,  // 订单生命周期
   datetime              expiration=0,              // 订单过期时间
   const string          comment=""                 // 注释
   )
```

参数

*volume*
  [输入] 订单交易量。

*price*
  [输入] 订单价格。

*symbol=NULL*
  [输入] 订单品名。如果品名未指定，则使用当前品名。

*sl=0.0*
  [输入] 止损价位。

*tp=0.0*
  [输入] 止盈价位。

*type_time=ORDER_TIME_GTC*
  [输入] 订单生命周期（值为 ENUM_ORDER_TYPE_TIME 枚举）。

*expiration=0*
  [输入] 订单过期时间（仅用在 type_time=ORDER_TIME_SPECIFIED）。

*comment=""*
  [输入] 订单注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 SellLimit(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultOrder() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# SellStop

按照指定参数放置低空类型挂单（在低于市场现价的价位卖出）。

```
bool  SellStop(
   double                volume,                    // 订单交易量
   double                price,                     // 订单价格
   const string          symbol=NULL,               // 品名
   double                sl=0.0,                    // 止损价
   double                tp=0.0,                    // 止盈价
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,  // 订单生命周期
   datetime              expiration=0,              // 订单过期时间
   const string          comment=""                 // 注释
   )
```

参数

*volume*
[输入] 订单交易量。

*price*
[输入] 订单价格。

*symbol=NULL*
[输入] 订单品名。如果品名未指定，则使用当前品名。

*sl=0.0*
[输入] 止损价位。

*tp=0.0*
[输入] 止盈价位。

*type_time=ORDER_TIME_GTC*
[输入] 订单生命周期（值为 ENUM_ORDER_TYPE_TIME 枚举）。

*expiration=0*
[输入] 订单过期时间（仅用在 type_time=ORDER_TIME_SPECIFIED）。

*comment=""*
[输入] 订单注释。

返回值

true -结构检查成功情况下，否则 false。

注释

成功完成 SellStop(...) 方法并不意味着总是成功执行交易操作。有必要检查交易请求的结果（交易服务器 返回代码），即 ResultRetcode() 和 ResultOrder() 返回的代码。

© 2000-2025, MetaQuotes Ltd.

# Request

获取最后请求结构的拷贝。

```
void  Request(
   MqlTradeRequest&  request      // 目标结构
   ) const
```

参数

*request*

[输出] MqlTradeRequest 类型的结构引用。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# RequestAction

获取交易操作类型。

```
ENUM_TRADE_REQUEST_ACTIONS  RequestAction() const
```

返回值

最后一笔请求使用的交易操作类型。

© 2000-2025, MetaQuotes Ltd.

# RequestActionDescription

获取交易操作类型的字符串描述。

```
string  RequestActionDescription() const
```

返回值

最后一笔请求使用的交易操作类型的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# RequestMagic

获取智能交易的魔幻数字。

```
ulong  RequestMagic() const
```

返回值

最后一笔请求使用的智能交易的魔幻数字。

© 2000-2025, MetaQuotes Ltd.

# RequestOrder

获取最后一笔请求使用的订单号。

```
ulong  RequestOrder() const
```

返回值

最后一笔请求使用的订单号。

© 2000-2025, MetaQuotes Ltd.

# RequestSymbol

获取最后一笔请求使用的品种名称。

```
string  RequestSymbol() const
```

返回值

最后一笔请求使用的品种名称。

© 2000-2025, MetaQuotes Ltd.

# RequestVolume

获取最后一笔请求使用的交易量 (手数为单位)。

```
double  RequestVolume() const
```

返回值

最后一笔请求使用的交易量 (手数为单位)。

© 2000-2025, MetaQuotes Ltd.

# RequestPrice

获取最后一笔请求使用的价格。

```
double  RequestPrice() const
```

返回值

最后一笔请求使用的订单价格。

© 2000-2025, MetaQuotes Ltd.

# RequestStopLimit

获取最后一笔请求使用的 StopLimit 类型挂单的价位。

```
double   RequestStoplimit() const
```

返回值

最后一笔请求使用的 StopLimit 类型挂单的价位。

© 2000-2025, MetaQuotes Ltd.

# RequestSL

获取最后一笔请求使用的订单止损价。

```
double  RequestSL() const
```

返回值

最后一笔请求使用的订单止损价。

© 2000-2025, MetaQuotes Ltd.

# RequestTP

获取最后一笔请求使用的订单止盈价。

```
double  RequestTP() const
```

返回值

最后一笔请求使用的订单止盈价。

© 2000-2025, MetaQuotes Ltd.

# RequestDeviation

获取最后一笔请求使用的订单价格滑点。

```
ulong  RequestDeviation() const
```

返回值

最后一笔请求使用的订单价格滑点。

© 2000-2025, MetaQuotes Ltd.

# RequestType

获取最后一笔请求使用的订单类型。

```
ENUM_ORDER_TYPE  RequestType() const
```

返回值

最后一笔请求使用的订单类型（值为 ENUM_ORDER_TYPE 枚举）。

© 2000-2025, MetaQuotes Ltd.

# RequestTypeDescription

获取最后一笔请求使用的订单类型的字符串描述。

```
string  RequestTypeDescription() const
```

返回值

最后一笔请求使用的订单类型的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# RequestTypeFilling

获取最后一笔请求使用的订单填充类型。

```
ENUM_ORDER_TYPE_FILLING  RequestTypeFilling() const
```

返回值

最后一笔请求使用的订单填充类型（值为 ENUM_ORDER_TYPE_FILLING）。

© 2000-2025, MetaQuotes Ltd.

# RequestTypeFillingDescription

获取最后一笔请求使用的订单填充类型的字符串描述。

```
string  RequestTypeFillingDescription() const
```

返回值

最后一笔请求使用的订单填充类型的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# RequestTypeTime

获取最后一笔请求使用的订单有效期。

```
ENUM_ORDER_TYPE_TIME  RequestTypeTime() const
```

返回值

最后一笔请求使用的订单有效期（值为 ENUM_ORDER_TYPE_TIME 枚举）。

© 2000-2025, MetaQuotes Ltd.

# RequestTypeTimeDescription

获取最后一笔请求订单使用的有效期的字符串描述。

```
string  RequestTypeTimeDescription() const
```

返回值

获取最后一笔请求使用的订单有效期的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# RequestExpiration

获取最后一笔请求使用的订单过期时间。

```
datetime  RequestExpiration() const
```

返回值

最后一笔请求使用的订单过期时间。

© 2000-2025, MetaQuotes Ltd.

# RequestComment

获取最后一笔请求使用的订单注释。

```
string  RequestComment() const
```

返回值

最后一笔请求使用的订单注释。

© 2000-2025, MetaQuotes Ltd.

# RequestPosition

获得持仓价格。

```
ulong  RequestPosition() const
```

返回值

用于最后请求的持仓价格。

© 2000-2025, MetaQuotes Ltd.

# RequestPositionBy

获得反向持仓价格。

```
ulong  RequestPositionBy() const
```

返回值

用于最后请求的反向持仓价格。

© 2000-2025, MetaQuotes Ltd.

# Result

获取最后请求结果结构的拷贝。

```
void  Result(
   MqlTradeResult&  result      // 引用
   ) const
```

参数

*result*

[输出] MqlTradeResult 类型的结构引用。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ResultRetcode

获取请求结果的代码。

```
uint  ResultRetcode() const
```

返回值

请求结果的 Code。

© 2000-2025, MetaQuotes Ltd.

# ResultRetcodeDescription

获取请求结果代码的文本。

```
string  ResultRetcodeDescription() const
```

返回值

最后一笔请求结果代码的文本。

© 2000-2025, MetaQuotes Ltd.

# ResultDeal

获取成交单单号。

```
ulong  ResultDeal() const
```

返回值

如果成交单已执行，则为成交单号。

© 2000-2025, MetaQuotes Ltd.

# ResultOrder

获取订单单号。

```
ulong  ResultOrder() const
```

返回值

如果订单已放置，则为订单号。

© 2000-2025, MetaQuotes Ltd.

# ResultVolume

获取成交单或订单的交易量。

```
double  ResultVolume() const
```

返回值

成交单或订单的交易量。

© 2000-2025, MetaQuotes Ltd.

# ResultPrice

获取券商确认的价格。

```
double  ResultPrice() const
```

返回值

券商确认的价格。

© 2000-2025, MetaQuotes Ltd.

# ResultBid

当前卖出价格（报价）。

```
double  ResultBid() const
```

返回值

当前卖出价格（报价）。

© 2000-2025, MetaQuotes Ltd.

# ResultAsk

获取当前买入价格 (报价)。

```
double  ResultAsk() const
```

返回值

当前买入价格 (报价)。

© 2000-2025, MetaQuotes Ltd.

# ResultComment

获取券商注释。

```
string  ResultComment() const
```

返回值

券商对交易的注释。

© 2000-2025, MetaQuotes Ltd.

# CheckResult

获取最后请求检查结果的结构拷贝

```
void  CheckResult(
   MqlTradeCheckResult&  check_result      // 引用
   ) const
```

参数

*check_result*

[输出] 目标 MqlTradeCheckResult 类型的结构引用。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CheckResultRetcode

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段的值。

```
uint  CheckResultRetcode() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段（错误代码）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultRetcodeDescription

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段值的字符串描述。

```
string ResultRetcodeDescription() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 retcode 字段（错误代码）值的字符串描述。

© 2000-2025, MetaQuotes Ltd.

# CheckResultBalance

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 balance 字段的值。

```
double   CheckResultBalance() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 balance 字段（交易操作执行后）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultEquity

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 equity 字段的值。

```
double   CheckResultEquity() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 equity 字段（交易操作执行后）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultProfit

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 profit 字段的值。

```
double   CheckResultProfit() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 profit 字段（交易操作执行后）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultMargin

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin 字段的值。

```
double   CheckResultMargin() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 margin 字段（操作所需的保证金）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultMarginFree

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_free 字段的值。

```
double  CheckResultMarginFree() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_free 字段（交易操作执行后的可用保证金）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultMarginLevel

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_level字段的值。

```
double  CheckResultMarginLevel() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 margin_level字段（交易操作执行后设置）的值。

© 2000-2025, MetaQuotes Ltd.

# CheckResultComment

获取检查请求正确性时填充的 MqlTradeCheckResult 类型 comment 字段的值。

```
string  CheckResultComment() const
```

返回值

检查请求正确性时填充的 MqlTradeCheckResult 类型 comment 字段（回复代码，错误描述）的值。

© 2000-2025, MetaQuotes Ltd.

# PrintRequest

在日志里输出最后一笔请求的参数。

```
void  PrintRequest() const
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# PrintResult

在日志里输出最后一笔请求的结果。

```
void  PrintResult() const
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# FormatRequest

准备最后一笔请求参数的格式字符串。

```
string  FormatRequest(
   string&                  str,          // 目标字符串
   const MqlTradeRequest&   request       // 请求
   ) const
```

参数

*str*

[输入] 目标字符串，通过引用传递。

*request*

[输入] 最后一笔 MqlTradeRequest 类型的带参数请求结构。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# FormatRequestResult

准备最后一笔请求执行结果的格式字符串。

```
string  FormatRequestResult(
   string&                 str,          // 字符串
   const MqlTradeRequest&  request,      // 请求结构
   const MqlTradeResult&   result        // 结果结构
   ) const
```

参数

*str*

[输入] 目标字符串，通过引用传递。

*request*

[输入] 最后一笔 MqlTradeRequest 类型的带参数请求结构。

*result*

[输入] 最后一笔请求的 MqlTradeResult 类型的结果结构。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CTerminalInfo

CTerminalInfo 类用于简便地访问 mql5 程序环境的属性。

## 描述

CTerminalInfo 类可供访问 mql5 程序环境的属性。

## 声明

```
class CTerminalInfo : public CObject
```

## 标称库文件

```
#include <Trade\TerminalInfo.mqh>
```

## 继承体系

CObject

CTerminalInfo

## 类方法分组

| 用于访问整数型属性的方法 | |
|---|---|
| Build | 获取客户终端集成构建的版本号 |
| IsConnected | 获取有关与交易服务器连接的信息 |
| IsDLLsAllowed | 获取有关使用 DLL 权限的信息 |
| IsTradeAllowed | 获取有关交易权限的信息 |
| IsEmailEnabled | 获取发送电子邮件至 SMTP 服务器的权限和登录信息，需在终端设置里指定 |
| IsFtpEnabled | 获取发送交易报告至 FTP 服务器的权限和登录信息，需在终端设置里指定 |
| MaxBars | 获取有关在图表上最大柱线数量的信息 |
| CodePage | 获取有关客户终端语言代码页的信息 |
| CPUCores | 获取有关 CPU 核心的信息 |
| MemoryPhysical | 获取有关物理内存的信息（以兆字节为单位） |
| MemoryTotal | 获取有关提供给终端/代理进程的总内存的信息 （以兆字节为单位） |
| MemoryAvailable | 获取有关可供给终端/代理进程的自由内存的信息 （以兆字节为单位） |
| MemoryUsed | 获取有关终端/代理进程使用内存的信息（以兆字节为单位） |

© 2000-2025, MetaQuotes Ltd.

| 用于访问整数型属性的方法 | |
|---|---|
| IsX64 | 获取有关客户终端类型的信息 (32/64 位) |
| OpenCLSupport | 获取有关显示卡支持的 OpenCL 版本信息 |
| DiskSpace | 获取有关自由磁盘空间的信息 (以兆字节为单位) |
| 用于访问字符串型属性的方法 | |
| Language | 获取客户终端语言 |
| 名称 | 获取客户终端名称 |
| Company | 获取客户终端所属公司 |
| Path | 获取客户终端文件夹 |
| DataPath | 获取客户终端数据文件夹 |
| CommonDataPath | 获取安装在电脑上的客户终端公用数据文件夹 |
| 访问 MQL5 API 函数 | |
| InfoInteger | 获取整数类型属性的值 |
| InfoString | 获取字符串类型属性的值 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Build

获取客户终端集成构建的版本号。

```
int  CBuild() const
```

返回值

客户终端集成构建的版本号。

注释

为获取集成构建的版本号，使用 TerminalInfoInteger() 函数（TERMINAL_BUILD 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsConnected

获取有关与交易服务器连接的信息。

```
bool  IsConnected() const
```

返回值

true，如果终端已与交易服务器连接，否则 false。

注释

为获取连接状态，使用 TerminalInfoInteger() 函数（TERMINAL_CONNECTED 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsDLLsAllowed

获取有关使用 DLL 权限的信息。

```
bool  IsDLLsAllowed() const
```

返回值

true，如果允许使用 DLL，否则 false。

注释

为获取 DLL 使用权限，使用 TerminalInfoInteger() 函数（TERMINAL_DLLS_ALLOWED 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsTradeAllowed

获取有关交易权限的信息。

```
bool  IsTradeAllowed() const
```

返回值

true 如果交易允许，否则 false。

注释

为获取交易权限，使用 TerminalInfoInteger() 函数（TERMINAL_TRADE_ALLOWED 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsEmailEnabled

获取发送电子邮件至 SMTP 服务器的权限和登录信息，需在终端设置里指定。

```
bool  IsEmailEnabled() const
```

返回值

true，如果发送 e-mails 允许，否则 false。

注释

为获取发送 e-mails 权限，使用 TerminalInfoInteger() 函数（TERMINAL_EMAIL_ENABLED 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsFtpEnabled

获取发送交易报告至 FTP 服务器的权限和登录信息，需在终端设置里指定。

```
bool  IsFtpEnabled() const
```

返回值

true，如果发送交易报告至 FTP 服务器允许，否则 false。

注释

为获取发送交易报告至 FTP 服务器权限，使用 TerminalInfoInteger() 函数 (TERMINAL_FTP_ENABLED 属性)。

© 2000-2025, MetaQuotes Ltd.

# MaxBars

获取客户端设置里指定的图表最大柱线数量。

```
int  MaxBars() const
```

返回值

图表最大柱线数量。

注释

为获取图表最大柱线数量，使用 TerminalInfoInteger() 函数（TERMINAL_MAXBARS 属性）。

© 2000-2025, MetaQuotes Ltd.

# CodePage

获取有关客户终端语言代码页的信息。

```
int  CodePage() const
```

返回值

客户终端语言代码页的信息。

注释

为获取语言代码页，使用 TerminalInfoInteger() 函数（TERMINAL_CODEPAGE 属性）。

© 2000-2025, MetaQuotes Ltd.

# CPUCores

获取有关系统中 CPU 核心数量的信息。

```
int  CPUCores() const
```

返回值

系统中 CPU 核心数量。

注释

为获取系统中 CPU 核心数量，使用 TerminalInfoInteger() 函数（TERMINAL_CPU_CORES 属性）。

© 2000-2025, MetaQuotes Ltd.

# MemoryPhysical

获取有关物理内存的信息（以兆字节为单位）。

```
int  MemoryPhysical() const
```

返回值

物理内存（以兆字节为单位）。<se

注释

为获取物理内存，使用 TerminalInfoInteger() 函数（TERMINAL_MEMORY_PHYSICAL 属性）。

© 2000-2025, MetaQuotes Ltd.

# MemoryTotal

获取有关提供给终端/代理的总内存的信息（以兆字节为单位）。

```
int  MemoryTotal() const
```

返回值

可供终端/代理使用的总内存（以兆字节为单位）。

注释

为获取总内存，使用 TerminalInfoInteger() 函数（TERMINAL_MEMORY_TOTAL 属性）。

© 2000-2025, MetaQuotes Ltd.

# MemoryAvailable

获取有关提供给终端/代理的自由内存的信息（以兆字节为单位）。

```
int  MemoryTotal() const
```

返回值

可供终端/代理使用的自由内存。

注释

为获取自由内存，使用 TerminalInfoInteger() 函数（TERMINAL_MEMORY_TOTAL 属性）。

© 2000-2025, MetaQuotes Ltd.

# MemoryUsed

获取有关终端/代理进程使用内存的信息（以兆字节为单位）。

```
int  MemoryUsed() const
```

返回值

终端/代理进程使用内存（以兆字节为单位）。

注释

为获取终端使用内存，使用 TerminalInfoInteger() 函数（TERMINAL_MEMORY_USED 属性）。

© 2000-2025, MetaQuotes Ltd.

# IsX64

获取有关客户终端类型的信息 (32/64 位)。

```
bool  IsX64() const
```

返回值

true，如果使用 64-位版本，否则 false。

注释

为获取客户终端类型，使用 TerminalInfoInteger() 函数 (TERMINAL_X64 属性)。

© 2000-2025, MetaQuotes Ltd.

# OpenCLSupport

获取有关显示卡支持的 OpenCL 版本信息。

```
int  OpenCLSupport() const
```

返回值

返回值有以下形式：0x00010002 = "1.2"。返回 0 意味着 OpenCL 不支持。

注释

为获取 OpenCL版本，使用 TerminalInfoInteger() 函数（TERMINAL_OPENCL_SUPPORT 属性）。

© 2000-2025, MetaQuotes Ltd.

# DiskSpace

获取有关提供给终端/代理的自由硬盘空间的信息 (以兆字节为单位)。

```
int  MDiskSpace() const
```

返回值

提供给终端/代理的自由硬盘空间的信息 (以兆字节为单位)。

注释

为获取自由硬盘空间，使用 TerminalInfoInteger() 函数 (TERMINAL_DISK_SPACE 属性)。

© 2000-2025, MetaQuotes Ltd.

# Language

获取有关客户终端语言的信息。

```
string  Language() const
```

返回值

客户终端使用的语言。

注释

为获取语言，使用 TerminalInfoString() 函数（TERMINAL_LANGUAGE 属性）。

© 2000-2025, MetaQuotes Ltd.

# 名称

获取有关客户终端名称信息。

```
string  Name() const
```

返回值

客户终端名称信息。

注释

为获取客户终端名称，使用 TerminalInfoString() 函数（TERMINAL_NAME 属性）。

© 2000-2025, MetaQuotes Ltd.

# Company

获取有关经纪商名称的信息。

```
string  Company() const
```

返回值

经纪商名称。

注释

为获取经纪商名称，使用 TerminalInfoString() 函数（TERMINAL_COMPANY 属性）。

© 2000-2025, MetaQuotes Ltd.

# Path

获取客户终端文件夹。

```
string  Path() const
```

返回值

客户终端文件夹。

注释

为获取终端数据文件夹，使用 TerminalInfoString() (TERMINAL_PATH 属性)。

© 2000-2025, MetaQuotes Ltd.

# DataPath

获取有关终端数据文件夹的信息。

```
string  DataPath() const
```

返回值

终端数据文件夹的信息。

注释

为获取终端数据文件夹的信息，使用 TerminalInfoString() 函数（TERMINAL_DATA_PATH 属性）。

© 2000-2025, MetaQuotes Ltd.

# CommonDataPath

获取安装在电脑上的客户终端公用数据文件夹。

```
string  CommonDataPath() const
```

返回值

公用数据文件夹。

注释

为获取公用数据文件夹，使用 TerminalInfoString() 函数（COMMON_DATA_PATH 属性）。

© 2000-2025, MetaQuotes Ltd.

# InfoInteger

返回 mql5 程序环境相应属性的值。

```
int  TerminalInfoInteger(
   int  property_id      // 属性标识符
   );
```

参数

*property_id*

[输入]属性标识符。其值可为 ENUM_TERMINAL_INFO_INTEGER枚举之一。

返回值

整型值。

注释

为获取属性值，使用 TerminalInfoInteger() 函数。

© 2000-2025, MetaQuotes Ltd.

# InfoString

函数返回 mql5 程序环境相应属性的值。属性必须是字符串类型。

```
string  TerminalInfoString(
   int  property_id      // 属性标识符
   );
```

参数

*property_id*

[输入]属性标识符。其值或许为 ENUM_TERMINAL_INFO_STRING枚举之一。

返回值

字符串型的值。

注释

为获取属性值，使用 TerminalInfoString() 函数。

© 2000-2025, MetaQuotes Ltd.

# 交易策略类

这章节包含操作创建和测试交易策略的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建 MQL5 的交易策略时，使用这些类，将节省大量时间。

MQL5 标准库（就交易策略而言）位于终端工作目录的 Include\Expert 文件夹。

| 基准类 | 描述 |
| --- | --- |
| CExpertBase | 交易策略类的基准类 |
| CExpert | 智能交易的基准类 |
| CExpertSignal | 交易信号类的基准类 |
| CExpertTrailing | 尾随停止类的基准类 |
| CExpertMoney | 资金管理类的基准类 |

| 交易信号类 | 描述 |
| --- | --- |
| CSignalAC | 该信号模块基于加速计振荡器指标的市场模型。 |
| CSignalAMA | 该信号模块基于自适应移动均线指标的市场模型。 |
| CSignalAO | 该信号模块基于动量振荡器指标的市场模型。 |
| CSignalBearsPower | 该信号模块基于空头力度振荡器的市场模型。 |
| CSignalBullsPower | 该信号模块基于多头力度振荡器的市场模型。 |
| CSignalCCI | 该信号模块基于商品通道指数振荡器的市场模型。 |
| CSignalDeM | 该信号模块基于 DeMarker 振荡器的市场模型。 |
| CSignalDEMA | 该信号模块基于双重指数移动均线指标的市场模型。 |
| CSignalEnvelopes | 该信号模块基于包络线指标的市场模型。 |
| CSignalFrAMA | 该信号模块基于分形自适应移动均线指标的市场模型。 |
| CSignalITF | 该模块以时间过滤信号 |
| CSignalMACD | 该信号模块基于 MACD 振荡器的市场模型。 |
| CSignalMA | 该信号模块基于移动均线指标的市场模型。 |
| CSignalSAR | 该信号模块基于抛物线转向指标的市场模型。 |
| CSignalRSI | 该信号模块基于相对强度指数振荡器的市场模型。 |
| CSignalRVI | 该信号模块基于相对能量指数振荡器的市场模型。 |
| CSignalStoch | 该信号模块基于随机振荡器的市场模型。 |
| CSignalTRIX | 该信号模块基于三重指数均线振荡器的市场模型。 |
| CSignalTEMA | 该信号模块基于双重指数移动均线指标的市场模型。 |

© 2000-2025, MetaQuotes Ltd.

| 交易信号类 | 描述 |
| --- | --- |
| CSignalWPR | 该信号模块基于威廉姆斯百分比范围振荡器的市场模型。 |

| 尾随停止类 | 描述 |
| --- | --- |
| CTrailingFixedPips | 该类实现的尾随停止算法基于固定点数 |
| CTrailingMA | 该类实现的尾随停止算法基于移动均线指标值 |
| CTrailingNone | 一个存根类，它不使用任何尾随停止算法 |
| CTrailingPSAR | 该类实现的尾随停止算法基于抛物线转向指标值 |

| 资金管理类 | 描述 |
| --- | --- |
| CMoneyFixedLot | 该类具有的算法基于预定义固定手数进行交易 |
| CMoneyFixedMargin | 该类具有的算法基于预定义固定保证金进行交易 |
| CMoneyFixedRisk | 该类具有的算法基于预定义风险进行交易 |
| CMoneyNone | 该类具有的算法基于最小允许手数进行交易 |
| CMoneySizeOptimized | 该类具有的算法基于变化手数进行交易，变化手数依据之前成交的结果。 |

© 2000-2025, MetaQuotes Ltd.

# 智能交易基准类

这章节包含操作创建和测试交易策略的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建 MQL5 的交易策略时，使用这些类，将节省大量时间。

MQL5 标准库（就交易策略而言）位于终端工作目录的 Include\Expert 文件夹。

| 类 | 描述 |
| --- | --- |
| CExpertBase | 交易策略类的基准类 |
| CExpert | 智能交易的基准类 |
| CExpertSignal | 交易信号类的基准类 |
| CExpertTrailing | 尾随停止类的基准类 |
| CExpertMoney | 资金管理类的基准类 |

© 2000-2025, MetaQuotes Ltd.

# CExpertBase

CExpertBase 类是 CExpert 类和所有交易策略类的基准类。

## 描述

CExpertBase 提供的数据和方法，是所有智能交易对象共有的。

## 声明

```
class CExpertBase : public CObject
```

## 标称库文件

```
#include <Expert\ExpertBase.mqh>
```

## 继承体系

CObject

CExpertBase

直接分支

CExpert, CExpertMoney, CExpertSignal, CExpertTrailing

## 类方法

## 公有方法：

| 初始化 | |
|---|---|
| virtual Init | 类实例初始化方法 |
| virtual ValidationSettings | 验证设置 |
| 参数 | |
| Symbol | 设置品名 |
| Period | 设置时间帧 |
| Magic | 设置智能交易 ID |
| 指标和时间序列 | |
| virtual SetPriceSeries | 设置外部时间序列指针（价格序列） |
| virtual SetOtherSeries | 设置外部时间序列指针（非价格序列） |
| virtual InitIndicators | 初始化指标和时间序列 |
| 访问保护数据 | |
| InitPhase | 获取对象初始化的当前阶段 |
| TrendType | 设置趋势类型 |

© 2000-2025, MetaQuotes Ltd.

| 初始化 | |
|---|---|
| UsedSeries | 获取时间序列使用的位掩码 |
| EveryTick | 设置"每笔报价"标志 |
| 访问时间序列 | |
| Open | 按照索引获取开盘价时间序列的元素 |
| High | 按照索引获取最高价时间序列的元素 |
| Low | 按照索引获取最低价时间序列的元素 |
| Close | 按照索引获取收盘价时间序列的元素 |
| Spread | 按照索引获取点差时间序列的元素 |
| Time | 按照索引获取时间轴时间序列的元素 |
| TickVolume | 按照索引获取报价交易量时间序列的元素 |
| RealVolume | 按照索引获取实际交易量时间序列的元素 |

## 保护方法：

| 时间序列初始化 | |
|---|---|
| InitOpen | 开盘价时间序列的初始化方法 |
| InitHigh | 最高价时间序列的初始化方法 |
| InitLow | 最低价时间序列的初始化方法 |
| InitClose | 收盘价时间序列的初始化方法 |
| InitSpread | 点差时间序列的初始化方法 |
| InitTime | 时间轴时间序列的初始化方法 |
| InitTickVolume | 报价交易量时间序列的初始化方法 |
| InitRealVolume | 实际交易量时间序列的初始化方法 |
| 服务方法 | |
| virtual PriceLevelUnit | 获取价格单位 |
| virtual StartIndex | 获取起始分析的柱线索引。 |
| virtual CompareMagic | 智能交易 ID 与指定值比较 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# InitPhase

获取对象初始化的当前阶段。

```
ENUM_INIT_PHASE  InitPhase()
```

返回值

对象初始化的当前阶段。

注释

对象的初始化包括几个阶段：

1. 初始化开始。

   -开始 -构造结束之后
   -结束 -在成功完成 Init(...) 方法之后。
   -允许 -调用 Init(...) 方法
   -不允许 -调用 ValidationSettings() 方法和其它初始化方法

2. 参数设置阶段。在此阶段您需要设置所有对象的参数，用于创建指标。

   -开始 -在成功完成 Init(...) 方法之后
   -结束 -在成功完成 ValidationSettings() 方法之后
   -允许 -调用 Symbol(...) 和 Period(...) 方法
   -不允许 -调用 Init(...), SetPriceSeries(...), SetOtherSeries(...) 和 InitIndicators(...) 方法

3. 检查参数。

   -开始 -在成功完成 ValidationSettings() 方法之后
   -结束 -在成功完成 InitIndicators(...) 方法之后
   -允许 -调用 Symbol(...), Period(...) 和 InitIndicators(...) 方法
   -不允许 -调用所有其它初始化方法

4. 初始化结束。

   -开始 -在成功完成 InitIndicators(...) 方法之后
   -不允许 -调用初始化方法

© 2000-2025, MetaQuotes Ltd.

# TrendType

设置趋势类型。

```
void  TrendType(
   M_TYPE_TREND    value          // 新值
   )
```

参数

*value*

[输入] 趋势类型新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# UsedSeries

获取时间序列使用的位掩码。

```
int  UsedSeries()
```

返回值

使用的时间序列列表作为位掩码。

注释

如果位掩码被设置，则相应的时间序列被使用，如果未设置，则时间序列不使用。

数码位与时间序列的对应：

位 0 - 开盘价时间序列，
位 1 - 最高价时间序列，
位t2 - 最低价时间序列，
位 3 - 收盘价时间序列，
位 4 - 点差时间序列，
位 5 - 时间轴时间序列，
位 6 - 报价交易量时间序列，
位 7 - 实际交易量时间序列。

© 2000-2025, MetaQuotes Ltd.

# EveryTick

设置"每笔报价"标志。

```
void  EveryTick(
   bool    value          // 标志
   )
```

参数

*value*

[输入] 新的标志值。

返回值

无。

注释

如果标志未设置，仅在交易品种工作时间帧生成新柱线时调用处理方法。

© 2000-2025, MetaQuotes Ltd.

# Open

按照索引获取开盘价时间序列的元素。

```
double  Open(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回开盘价时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# High

按照索引获取最高价时间序列的元素。

```
double  High(
   int     ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回最高价时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# Low

按照索引获取最低价时间序列的元素。

```
double  Low(
   int     ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回最低价时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# Close

按照索引获取收盘价时间序列的元素。

```
double  Close(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回收盘价时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# Spread

按照索引获取点差时间序列的元素。

```
double  Spread(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回点差时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# Time

按照索引获取时间轴时间序列的元素。

```
datetime  Time(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回时间轴时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# TickVolume

按照索引获取报价交易量时间序列的元素。

```
long  TickVolume(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回报价交易量时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# RealVolume

按照索引获取实际交易量时间序列的元素。

```
long  RealVolume(
   int    ind          // 索引
   )
```

参数

*ind*

[输入] 元素索引。

返回值

如果成功，它返回实际交易量时间序列里指定索引处的元素值，否则它返回 EMPTY_VALUE。

注释

在两种情况下返回 EMPTY_VALUE:

1. 时间序列不可用 (相应位未设置)。
2. 元素索引超界。

© 2000-2025, MetaQuotes Ltd.

# Init

初始化交易对象。

```
bool  Init(
   CSymbolInfo       symbol,    // 品名
   ENUM_TIMEFRAMES   period,    // 时间帧
   double            point      // 点数
   )
```

参数

*symbol*

[输入] 用于访问品种信息的 CSymbolInfo 类对象的指针。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*point*

[输入] 2/4-小数点之后有效位数的 "权重"。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Symbol

设置品名。

```
bool  Symbol(
   string    name     // 品名
   )
```

参数

*name*

[输入] 品名。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名则设置工作品名是必要的，与初始化时定义的品名不同。

© 2000-2025, MetaQuotes Ltd.

# Period

设置时间帧。

```
bool  Period(
   ENUM_TIMEFRAMES  value     // 时间帧
   )
```

参数

*value*

[输入] 时间帧。

返回值

true 如果成功，否则 false。

注释

如果对象使用了时间帧则设置工作时间帧是必要的，与初始化时定义的时间帧不同。

© 2000-2025, MetaQuotes Ltd.

# Magic

设置智能交易 ID。

```
void  Magic(
   ulong  value      // 魔幻数字
   )
```

参数

*value*

[输入] 智能交易 ID。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# SetPriceSeries

设置外部价格序列指针。

```
virtual bool  SetPriceSeries(
   CiOpen*     open,         // 指针
   CiHigh*     high,         // 指针
   CiLow*      low,          // 指针
   CiClose*    close         // 指针
   )
```

参数

*open*

[输入] 开盘价序列的指针。

*high*

[输入] 最高价序列的指针。

*low*

[输入] 最低价序列的指针。

*close*

[输入] 收盘价序列的指针。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名或时间帧则设置外部序列（价格序列）的指针是必要的，与初始化时定义的品名或时间帧不同。

© 2000-2025, MetaQuotes Ltd.

# SetOtherSeries

设置外部非价格序列指针。

```
virtual bool  SetOtherSeries(
   CiSpread*        spread,         // 指针
   CiTime*          time,           // 指针
   CiTickVolume*    tick_volume,    // 指针
   CiRealVolume*    real_volume     // 指针
   )
```

参数

*spread*

[输入] 点差序列的指针。

*time*

[输入] 时间轴序列的指针。

*tick_volume*

[输入] 报价交易量序列的指针。

*real_volume*

[输入] 实际交易量序列的指针。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名或时间帧则设置外部序列（非价格序列）的指针是必要的，与初始化时定义的品名或时间帧不同。

© 2000-2025, MetaQuotes Ltd.

# InitIndicators

初始化所有指标和时间序列。

```
virtual bool  InitIndicators(
   CIndicators*  indicators=NULL    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名或时间帧则时间序列被初始化，与初始化时定义的品名或时间帧不同。

© 2000-2025, MetaQuotes Ltd.

# InitOpen

初始化开盘价时间序列。

```
bool  InitOpen(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

开盘价时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitHigh

初始化最高价时间序列。

```
bool  InitHigh(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

最高价时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitLow

初始化最低价时间序列。

```
bool  InitLow(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

最低价时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitClose

初始化收盘价时间序列。

```
bool  InitClose(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

收盘价时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitSpread

初始化点差时间序列。

```
bool  InitSpread(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

点差时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitTime

初始化时间轴时间序列。

```
bool  InitTime(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

时间轴时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitTickVolume

初始化报价交易量时间序列。

```
bool  InitTickVolume(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

报价交易量时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# InitRealVolume

初始化实际交易量时间序列。

```
bool  InitRealVolume(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

实际交易量时间序列仅在智能交易使用品名/时间帧时被初始化，与初始化时定义的品名/时间帧不同 (时间帧会在以后使用)。

© 2000-2025, MetaQuotes Ltd.

# PriceLevelUnit

获取价格单位。

```
virtual double  PriceLevelUnit()
```

返回值

价格单位的值。

注释

基准类方法返回小数点之后有效位数 2/4 的 "权重"。

© 2000-2025, MetaQuotes Ltd.

# StartIndex

获取起始分析的柱线索引。

```
virtual int  StartIndex()
```

返回值

起始分析的柱线索引。

注释

方法返回 0，若分析当前柱线的标志设置为 true（从当前柱线开始分析）。如果标志未设置，它返回 1（从最后完整的柱线开始分析）。

© 2000-2025, MetaQuotes Ltd.

# CompareMagic

智能交易 ID 与指定值比较。

```
virtual bool  CompareMagic(
   ulong  magic    // 比较值
   )
```

参数

*magic*

[输入] 比较值。

返回值

true - 如果相等，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CExpert

CExpert 是交易策略的基准类。它有内建算法用于操作时间序列和指标以及一套交易策略的虚方法。

如何使用它：

1. 准备策略的算法；
2. 创建您自己的继承 CExpert 的子类；
3. 在您的类中用您自己的算法覆盖虚方法。

## 描述

类CExpert是一套虚方法用来实现交易策略。

## 声明

```
class CExpert : public CExpertBase
```

## 标称库文件

```
#include <Expert\Expert.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpert

## 类方法

| 初始化 | |
|---|---|
| Init | 类实例初始化方法 |
| virtual InitSignal | 初始化交易信号对象 |
| virtual InitTrailing | 初始化尾随停止对象 |
| virtual InitMoney | 初始化资金管理对象 |
| virtual InitTrade | 初始化交易对象 |
| virtual ValidationSettings | 验证设置 |
| virtual InitIndicators | 初始化指标和时间序列 |
| virtual InitParameters | 参数初始化方法 |
| virtual Deinit | 类实例终结方法 |
| virtual DeinitSignal | 终结交易信号对象 |
| virtual DeinitTrailing | 终结尾随停止对象 |
| virtual DeinitMoney | 终结资金管理对象 |

© 2000-2025, MetaQuotes Ltd.

| 初始化 | |
|---|---|
| virtual DeinitTrade | 终结交易对象 |
| virtual DeinitIndicators | 终结指标和时间序列 |
| 参数 | |
| Magic | 设置智能交易 ID |
| MaxOrders | 获取/设置许可的最大订单额度 |
| OnTickProcess | 设置处理 "OnTick" 事件的标志 |
| OnTradeProcess | 设置处理 "OnTrade" 事件的标志 |
| OnTimerProcess | 设置处理 "OnTimer" 事件的标志 |
| OnChartEventProcess | 设置处理 "OnChartEvent" 事件的标志 |
| OnBookEventProcess | 设置处理 "OnBookEvent" 事件的标志 |
| 事件处理方法 | |
| OnTick | OnTick 事件处理器 |
| OnTrade | OnTrade 事件处理器 |
| OnTimer | OnTimer 事件处理器 |
| OnChartEvent | OnChartEvent 事件处理器 |
| OnBookEvent | OnBookEvent 事件处理器 |
| 更新方法 | |
| Refresh | 更新所有数据 |
| 处理 | |
| 处理 | 主要处理算法 |
| 入场方法 | |
| CheckOpen | 检查开仓条件 |
| CheckOpenLong | 检查开多头仓位的条件 |
| CheckOpenShort | 检查开空头仓位的条件 |
| OpenLong | 开多头仓位 |
| OpenShort | 开空头仓位 |
| 离场方法 | |
| CheckClose | 检查当前平仓条件 |
| CheckCloseLong | 检查平多头仓位的条件 |
| CheckCloseShort | 检查平空头仓位的条件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 初始化 | |
| CloseAll | 平所有持仓并删除所有订单 |
| Close | 平已开持仓 |
| CloseLong | 平多头持仓 |
| CloseShort | 平空头持仓 |
| 仓位反转方法 | |
| CheckReverse | 检查已开仓位的反转条件 |
| CheckReverseLong | 检查反转多头仓位的条件 |
| CheckReverseShort | 检查反转空头仓位的条件 |
| ReverseLong | 执行多头仓位反转 |
| ReverseShort | 执行空头仓位反转 |
| 仓位/订单尾随停止方法 | |
| CheckTrailingStop | 检查修改仓位参数的条件 |
| CheckTrailingStopLong | 检查多头仓位的尾随停止条件 |
| CheckTrailingStopShort | 检查空头仓位的尾随停止条件 |
| TrailingStopLong | 执行多头仓位的尾随停止 |
| TrailingStopShort | 执行空头仓位的尾随停止 |
| CheckTrailingOrderLong | 检查低多/高多挂单的尾随停止条件 |
| CheckTrailingOrderShort | 检查高空/低空挂单的尾随停止条件 |
| TrailingOrderLong | 检查低多/高多挂单的尾随停止 |
| TrailingOrderShort | 检查高空/低空挂单的尾随停止 |
| 订单删除方法 | |
| CheckDeleteOrderLong | 检查删除买挂单的条件 |
| CheckDeleteOrderShort | 检查删除卖挂单的条件 |
| DeleteOrders | 删除所有订单 |
| DeleteOrder | 删除高/低挂单 |
| DeleteOrderLong | 删除低多/高多挂单 |
| DeleteOrderShort | 删除高空/低空挂单 |
| 交易量方法 | |
| LotOpenLong | 获取多头操作交易量 |
| LotOpenShort | 获取空头操作交易量 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 初始化 | |
| LotReverse | 获取仓位反转操作交易量 |
| 交易历史方法 | |
| PrepareHistoryDate | 设置交易历史跟踪的开始日期 |
| HistoryPoint | 创建交易历史的检测点（保存仓位，订单，成交和历史订单的数量） |
| CheckTradeState | 当前状态和之前的保存记录进行比较， 并调用相应的事件处理器 |
| 事件标志 | |
| WaitEvent | 设置事件等待标志 |
| NoWaitEvent | 重置事件等待标志 |
| 交易事件处理方法 | |
| TradeEventPositionStopTake | "仓位止损/止盈触发"事件处理器 |
| TradeEventOrderTriggered | "挂单触发"事件处理器 |
| TradeEventPositionOpened | "开仓触发"事件处理器 |
| TradeEventPositionVolumeChanged | "持仓量改变"事件处理器 |
| TradeEventPositionModified | "持仓修改"事件处理器 |
| TradeEventPositionClosed | "平仓"事件处理器 |
| TradeEventOrderPlaced | "放置挂单"事件处理器 |
| TradeEventOrderModified | "挂单修改"事件处理器 |
| TradeEventOrderDeleted | "挂单删除"事件处理器 |
| TradeEventNotIdentified | 无标识事件处理器 |
| 服务方法 | |
| TimeframeAdd | 添加时间帧跟踪 |
| TimeframesFlags | 获取新柱线的标志指示时间帧 |
| SelectPosition | 选择一个工作持仓 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

© 2000-2025, MetaQuotes Ltd.

# Init

类实例初始化方法。

```
bool  Init(
   string               symbol,        // 品名
   ENUM_TIMEFRAMES      period,        // 时间帧
   bool                 every_tick,    // 标志
   ulong                magic          // 魔幻数字
   )
```

参数

*symbol*

[输入] 品名。

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

*every_tick*

[输入] 标志。

*magic*

[输入] 智能交易 ID (魔幻数字)。

返回值

true 如果成功，否则 false。

注释

如果 every_tick 设置为 to true，则 Processing() 方法在所操作品种的每次报价时都被调用。否则，方法 Processing() 仅在所操作品种的新柱线建立时被调用。

© 2000-2025, MetaQuotes Ltd.

# Magic

设置智能交易 ID (魔幻数字)。

```
void  Magic(
   ulong  value     // 新值
   )
```

参数

*value*

[输入] 智能交易 ID 的新值。

返回值

无。

注释

它可为以下类设置智能交易 ID (魔幻数字): Trade, Signal, Money, Trailing。

实现

```
//+------------------------------------------------------------------+
//| 为对象和其以来对象设置魔幻数字                                   |
//| 输入:  value - 魔幻数字新值。                                    |
//| 输出: 无。                                                       |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
void CExpert::Magic(ulong value)
  {
   if(m_trade!=NULL)    m_trade.SetExpertMagicNumber(value);
   if(m_signal!=NULL)   m_signal.Magic(value);
   if(m_money!=NULL)    m_money.Magic(value);
   if(m_trailing!=NULL) m_trailing.Magic(value);
//---
   CExpertBase::Magic(value);
  }
```

© 2000-2025, MetaQuotes Ltd.

# InitSignal

初始化尾随信号对象。

```
virtual bool  InitSignal(
   CExpertSignal*     signal=NULL,         // 指针
   )
```

参数

*signal*

[输入] 类CExpertSignal对象（或其后代）的指针。

返回值

true 如果成功，否则 false。

注释

如果 signal为 NULL，则 CExpertSignal类将被使用，它什么也不做。

© 2000-2025, MetaQuotes Ltd.

# InitTrailing

初始化尾随停止对象。

```
virtual bool  InitTrailing(
   CExpertTrailing*     trailing=NULL,         // 指针
   )
```

参数

*trailing*

[输入] 类CExpertTrailing对象（或其后代）的指针。

返回值

true 如果成功，否则 false。

注释

如果 trailing 为 NULL，则 ExpertTrailing 类将被使用，它什么也不做。

© 2000-2025, MetaQuotes Ltd.

# InitMoney

初始化资金管理对象。

```
virtual bool  InitMoney(
   CExpertMoney*     money=NULL,         // 指针
   )
```

参数

*money*

[输入] 类CExpertMoney对象（或其后代）的指针。

返回值

true 如果成功，否则 false。

注释

如果 money 为 NULL，则 CExpertMoney 类将被使用，它使用最小手数。

© 2000-2025, MetaQuotes Ltd.

# InitTrade

初始化交易对象。

```
virtual bool  InitTrade(
   ulong             magic,        // 魔幻数字
   CExpertTrade*     trade=NULL    // 指针
   )
```

参数

*magic*

[输入] 智能交易 ID (将被用于交易请求)。

*trade*

[输入] CExpertTrade 对象的指针。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Deinit

类实例终结方法。

```
virtual void Deinit()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnTickProcess

设置处理 OnTick 事件的标志。

```
void  OnTickOProcess(
   bool     value        // 标志
   )
```

参数

*value*

[输入] <t1>处理 </t1><li2>OnTick</li2> 事件的标志。

返回值

无。

注释

如果标志为 true，则 OnTick 事件被处理，省缺，标志设置为 false。

© 2000-2025, MetaQuotes Ltd.

# OnTradeProcess

设置处理 OnTrade 事件的标志。

```
void  OnTradeProcess(
   bool     value        // 标志
   )
```

参数

*value*

[输入] 处理 OnTrade 事件的标志。

返回值

无。

注释

如果标志为 true，则 OnTrade 事件被处理，省缺，标志设置为 false。

© 2000-2025, MetaQuotes Ltd.

# OnTimerProcess

设置处理 OnTimer 事件的标志。

```
void  OnTimerProcess(
   bool     value        // 标志
   )
```

参数

*value*

[输入] 处理 OnTimer 事件的标志。

返回值

无。

注释

如果标志为 true，则 OnTimer 事件被处理，省缺，标志设置为 false。

© 2000-2025, MetaQuotes Ltd.

# OnChartEventProcess

设置处理 OnChartEvent 事件的标志。

```
void  OnChartEventProcess(
   bool     value        // 标志
   )
```

参数

*value*

[输入] 处理 OnChartEvent 事件的标志。

返回值

无。

注释

如果标志为 true，则 OnChartEvent 事件被处理，省缺，标志设置为 false。

© 2000-2025, MetaQuotes Ltd.

# OnBookEventProcess

设置处理 OnBookEvent 事件的标志。

```
void  OnChartEventProcess(
   bool     value        // 标志
   )
```

参数

*value*

[输入] 处理 OnBookEvent 事件的标志。

返回值

无。

注释

如果标志为 true，则 OnBookEvent 事件被处理，省缺，标志设置为 false。

© 2000-2025, MetaQuotes Ltd.

## MaxOrders (Get方法)

获取许可的最大订单额度。

```
int  MaxOrders()
```

返回值

许可的最大订单额度。

## MaxOrders (Set方法)

设置许可的最大订单额度。

```
void  MaxOrders(
   int      max_orders        // 新值
   )
```

参数

*max_orders*

[输入] 许可的最大订单额度新值。

返回值

无。

注释

省缺，许可的最大订单额度 = 1。

© 2000-2025, MetaQuotes Ltd.

# Signal

获取交易信号对象指针。

```
CExpertSignal*  Signal() const
```

返回值

交易信号对象指针。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

它检查所有智能交易对象的设置。

© 2000-2025, MetaQuotes Ltd.

# InitIndicators

初始化所有指标和时间序列。

```
virtual bool  InitIndicators(
   CIndicators*  indicators=NULL    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名或时间帧则时间序列被初始化，与初始化时定义的品名或时间帧不同。它持续调用交易信号、尾随停止、资金管理对象的 InitIndicators() 虚方法。

© 2000-2025, MetaQuotes Ltd.

# OnTick

事件 OnTick 的事件处理器。

```
virtual void  OnTick()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnTrade

事件 OnTrade 的事件处理器。

```
virtual void  OnTrade()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnTimer

事件 OnTimer 的事件处理器。

```
virtual void  OnTimer()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnChartEvent

事件 OnChartEvent 的事件处理器。

```
virtual void  OnChartEvent(
   const int     id,       // 事件 id
   const long&   lparam,   // 长整数型参数
   const double  dparam,   // 双精度型参数
   const string  sparam    // 字符串型参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 长整数型事件参数。

*dparam*

[输入] 双精度型事件参数。

*sparam*

[输入] 字符串型事件参数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnBookEvent

事件 OnBookEvent 的事件处理器。

```
virtual void  OnBookEvent(
   const string&     symbol        // 事件符号
)
```

参数

*symbol*

[输入] 事件 OnBookEvent 的事件符号。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# InitParameters

初始化智能交易的参数。

```
virtual bool  InitParameters()
```

返回值

true 如果成功，否则 false。

注释

基准类CExpert 中的 InitParameters() 函数不做任何事情且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# DeinitTrade

终结交易对象。

```
virtual void  DeinitTrade()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DeinitSignal

终结交易信号对象。

```
virtual void  DeinitSignal()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DeinitTrailing

终结尾随停止对象。

```
virtual void  DeinitTrailing()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DeinitMoney

终结资金管理对象。

```
virtual void  DeinitMoney()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DeinitIndicators

终结指标和时间序列。

```
virtual void  DeinitIndicators()
```

返回值

无。

注释

它还终结所有指标和所有辅助对象的时间序列。

© 2000-2025, MetaQuotes Ltd.

# Refresh

更新所有数据。

```
virtual bool  Refresh()
```

返回值

true 如果需要处理未来的报价，否则 false。

注释

它可确定处理报价的需要。如果需要，它更新所有的报价、时间序列和指标的数据并返回 true。

实现

```
//+------------------------------------------------------------------+
//| 刷新数据用于处理                                                 |
//| 输入:  无。                                                      |
//| 输出: true-若成功, 否则 false。                                  |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::Refresh()
  {
   MqlDateTime time;
//--- 刷新率
   if(!m_symbol.RefreshRates()) return(false);
//--- 检查处理需要
   TimeToStruct(m_symbol.Time(),time);
   if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
      if((m_period_flags & TimeframesFlags(time))==0) return(false);
   m_last_tick_time=time;
//--- 刷新指标
   m_indicators.Refresh();
//--- ok
   return(true);
  }
```

© 2000-2025, MetaQuotes Ltd.

# 处理

主要处理算法。

```
virtual bool  Processing()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它进行以下步骤：

1. 检查该品种是否有已开仓位。若无已开仓位，跳过步骤 #2, #3 和 #4。
2. 检查已开仓位的反转条件 (CheckReverse() 方法)。如果仓位已被 "反转"，退出。
3. 检查平仓条件 (CheckClose() 方法)。如果仓位已平仓，跳过步骤 #4。
4. 检查修改仓位参数的条件 (CheckTrailingStop() 方法)。如果仓位参数已被修改，退出。
5. 检查该品种是否有已开挂单。若没有任何挂单，直接到步骤 #9。
6. 检查删除订单的条件 (CheckDeleteOrderLong() 对应多头挂单，或者 CheckDeleteOrderShort() 对应空头挂单)。如果订单已被删除，直接到步骤 #9。
7. 检查修改挂单参数的条件 (CheckTrailingOrderLong() 对应多头挂单，或者 CheckTrailingOrderShort() 对应空头挂单)。如果订单参数已被修改，退出。
8. 退出。
9. 检查开仓条件 (CheckOpen()) 方法)。

若您打算实现您自己的算法，您需要覆盖其子类的 Processing() 方法。

实现

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| 主函数                                                           |
//| 输入:  无。                                                      |
//| 输出: true-如果任何交易操作已处理, 否则 false。                  |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::Processing()
  {
//--- 检查是否开仓
   if(m_position.Select(m_symbol.Name()))
     {
      //--- 可以开仓
      //--- 检查反转仓位的可能性
      if(CheckReverse()) return(true);
      //--- 检查平仓/删除挂单的可能性
      if(!CheckClose())
        {
         //--- 检查修改仓位的可能性
         if(CheckTrailingStop()) return(true);
         //--- 返回无操作
         return(false);
        }
     }
//--- 检查是否放置挂单
   int total=OrdersTotal();
   if(total!=0)
     {
      for(int i=total-1;i>=0;i--)
        {
         m_order.SelectByIndex(i);
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         if(m_order.OrderType()==ORDER_TYPE_BUY_LIMIT || m_order.OrderType()==ORDER_TY
           {
            //--- 检查删除多头挂单能力
            if(CheckDeleteOrderLong()) return(true);
            //--- 检查修改多头挂单的可能性
            if(CheckTrailingOrderLong()) return(true);
           }
         else
           {
            //--- 检查删除空头挂单的可能性
            if(CheckDeleteOrderShort()) return(true);
            //--- 检查修改空头挂单的可能性
            if(CheckTrailingOrderShort()) return(true);
           }
         //--- 返回无操作
         return(false);
        }
     }
//--- 检查开仓/设置挂单可能性
   if(CheckOpen()) return(true);
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# SelectPosition

选择一个工作持仓。

```
void  SelectPosition()
```

返回值

No.

实施

```
//+------------------------------------------------------------------+
//| 持仓选择                                                         |
//| INPUT:  no.                                                      |
//| OUTPUT: no.                                                      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::SelectPosition(void)
  {
   bool res=false;
//---
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
      res=m_position.SelectByMagic(m_symbol.Name(),m_magic);
   else
      res=m_position.Select(m_symbol.Name());
//---
   return(res);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckOpen

检查开仓的条件。

```
virtual bool  CheckOpen()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查开多头仓位 (CheckOpenLong()) 和空头仓位 (CheckOpenShort()) 的条件。

实现

```
//+------------------------------------------------------------------+
//| 检查开仓或挂单                                                   |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckOpen()
  {
   if(CheckOpenLong())  return(true);
   if(CheckOpenShort()) return(true);
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

检查开多头仓位的条件。

```
virtual bool  CheckOpenLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查开多头仓位的条件（信号对象的 CheckOpenLong() 方法）并在必要时开多头仓位 (OpenLong() 方法)。

实现

```
//+------------------------------------------------------------------+
//| 检查开多头仓位或挂单                                             |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- 检查多头入场操作信号
   if(m_signal.CheckOpenLong(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenLong(price,sl,tp));
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

检查开空头仓位的条件。

```
virtual bool  CheckOpenShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查开空头仓位的条件（信号对象的 CheckOpenShort() 方法）并在必要时开空头仓位 (OpenShort() 方法)。

实现

```
//+------------------------------------------------------------------+
//| 检查开空头仓位或挂单                                             |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- 检查空头入场操作信号
   if(m_signal.CheckOpenShort(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenShort(price,sl,tp));
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# OpenLong

开多头仓位。

```
virtual bool  OpenLong(
   double    price,    // 价位
   double    sl,       // 止损
   double    tp        // 止盈
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

*tp*

[输入] 止损价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

它获取交易量（LotOpenLong(...) 方法）且若交易量不等于 0，则开多头仓位（Trade 对象的 Buy() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 开多头仓位或设置挂单                                             |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损,                                           |
//|          tp    - 止盈。                                          |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::OpenLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- 获取开仓手数
   double lot=LotOpenLong(price,sl);
//--- 检查开仓手数
   if(lot==0.0) return(false);
//---
   return(m_trade.Buy(lot,price,sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# OpenShort

开空头仓位。

```
virtual bool  OpenShort(
   double    price,    // 价位
   double    sl,       // 止损
   double    tp        // 止盈
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

*tp*

[输入] 止损价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

它获取交易量（LotOpenShort(...) 方法）且若交易量不等于 0，则开空头仓位（Trade 对象的 Sell() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 开多头仓位或设置挂单                                             |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损,                                           |
//|          tp    - 止盈。                                          |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::OpenShort(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- 获取开仓手数
   double lot=LotOpenShort(price,sl);
//--- 检查开仓手数
   if(lot==0.0) return(false);
//---
   return(m_trade.Sell(lot,price,sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckReverse

检查已开仓位的反转条件。

```
virtual bool  CheckReverse()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查多头仓位 (CheckReverseLong()) 和空头仓位 (CheckReverseShort()) 的反转条件 。

实现

```
//+------------------------------------------------------------------+
//| 检查持仓反转                                                     |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckReverse()
  {
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- 检查反转多头持仓的可能性
      if(CheckReverseLong())  return(true);
     }
   else
      //--- 检查反转空头持仓的可能性
      if(CheckReverseShort()) return(true);
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckReverseLong

检查反转多头仓位的条件。

```
virtual bool  CheckReverseLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查多头持仓的反转条件（信号对象的 CheckReverseLong() 方法）并在必要时执行当前多头持仓的反转操作（ReverseLong(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查多头持仓反转                                                 |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- 检查多头持仓反转操作信号
   if(m_signal.CheckReverseLong(price,sl,tp,expiration)) return(ReverseLong(price,sl,t
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckReverseShort

检查反转空头持仓的条件。

```
virtual bool  CheckReverseLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查空头持仓的反转条件（信号对象的 CheckReverseShort() 方法）并在必要时执行当前空头持仓的反转操作（ReverseShort() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查空头持仓反转                                                 |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- 检查空头持仓反转操作信号
   if(m_signal.CheckReverseShort(price,sl,tp,expiration)) return(ReverseShort(price,sl
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# ReverseLong

执行多头仓位反转。

```
virtual bool  ReverseLong(
   double    price,    // 价位
   double    sl,       // 止损
   double    tp        // 止盈
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

*tp*

[输入] 止损价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

它获取仓位反转交易量（LotReverse(...) 方法）且若交易量不等于 0，则执行多头仓位反转（Trade 对象的 Sell() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 多头仓位反转                                                     |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损,                                           |
//|          tp    - 止盈。                                          |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::ReverseLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- 获取反转手数
   double lot=LotReverse(sl);
//--- 检查手数
   if(lot==0.0) return(false);
//---
   return(m_trade.Sell(lot,price,sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# ReverseShort

执行空头仓位反转。

```
virtual bool  ReverseShort(
   double    price,    // 价位
   double    sl,       // 止损
   double    tp        // 止盈
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

*tp*

[输入] 止损价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

它获取仓位反转交易量（LotReverse(...) 方法）且若交易量不等于 0，则执行空头仓位反转（Trade 对象的 Buy() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 空头仓位反转                                                     |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损,                                           |
//|          tp    - 止盈。                                          |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::ReverseShort(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- 获取反转手数
   double lot=LotReverse(sl);
//--- 检查手数
   if(lot==0.0) return(false);
//---
   return(m_trade.Buy(lot,price,sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckClose

检查平仓条件。

```
virtual bool  CheckClose()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

1. 它检查智能交易的爆仓条件（资金管理对象的 CheckClose() 方法）。如果条件满足，它平仓且删除所有订单（CloseAll(...)）并退出。
2. 它检查多头和空头的平仓条件（CheckCloseLong() 或 CheckCloseShort() methods）且当仓位已平时，删除所有订单（DeleteOrders() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查平仓或挂单删除                                               |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckClose()
  {
   double lot;
//--- 在调用之前仓位必须已被选择
   if((lot=m_money.CheckClose(GetPointer(m_position)))!=0.0)
      return(CloseAll(lot));
//--- 检查持仓类型
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- 检查平多头持仓 / 删除多头挂单的可能性
      if(CheckCloseLong())
        {
         DeleteOrders();
         return(true);
        }
     }
   else
     {
      //--- 检查平空头持仓 / 删除空头挂单的可能性
      if(CheckCloseShort())
        {
         DeleteOrders();
         return(true);
        }
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckCloseLong

检查平多头仓位的条件。

```
virtual bool  CheckCloseLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查多头持仓的平仓条件（信号对象的 CheckCloseLong() 方法）且当满足时，已开仓位平仓（CloseLong(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查多头持仓平仓或挂单删除                                       |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseLong()
  {
   double price=EMPTY_VALUE;
//--- 检查多头平仓操作
   if(m_signal.CheckCloseLong(price))
      return(CloseLong(price));
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckCloseShort

检查平空头仓位的条件。

```
virtual bool  CheckCloseShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查空头持仓的平仓条件（信号对象的 CheckCloseShort() 方法）且当满足时，已开仓位平仓（CloseShort() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查空头持仓平仓或挂单删除                                       |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseShort()
  {
   double price=EMPTY_VALUE;
//--- 检查空头平仓操作
   if(m_signal.CheckCloseShort(price))
      return(CloseShort(price));
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CloseAll

它执行满仓的部分平仓。

```
virtual bool  CloseAll(
   double    lot    // 手数
   )
```

参数

*lot*

[输入] 减仓的手数。

返回值

true 如果交易操作已被执行，否则 false。

注释

在 "单边持仓"模式，使用CExpertTrade::Buy或CExpertTrade::Sell 类函数来平仓。在 "锁仓持仓"模式，则使用CTrade::PositionClose 类函数，它也可以用于单边模式下的账户。DeleteOrders()类函数用于删除订单。

实现

```
//+------------------------------------------------------------------+
//| 平仓并删除订单                                                   |
//| 输入:  手数 - 平仓交易量。                                       |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CloseAll(double lot)
  {
   bool result;
//--- 检查平仓操作
   if(m_position.PositionType()==POSITION_TYPE_BUY) result=m_trade.Sell(lot,0,0,0);
   else                                             result=m_trade.Buy(lot,0,0,0);
   result|=DeleteOrders();
//---
   return(result);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Close

已开持仓平仓。

```
virtual bool  Close()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

平仓 (CTrade 类对象的 PositionClose() 方法)。

实现

```
//+------------------------------------------------------------------+
//| 平仓                                                             |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::Close()
  {
   return(m_trade.PositionClose(m_symbol.Name()));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CloseLong

多头持仓平仓。

```
virtual bool  CloseLong(
   double    price    // 价位
   )
```

参数

*price*

[输入] 价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

多头持仓平仓 (CTrade 类对象的 Sell(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 多头持仓平仓                                                     |
//| 输入:  价位 - 平仓价。                                           |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CloseLong(double price)
  {
   if(price==EMPTY_VALUE) return(false);
//---
   return(m_trade.Sell(m_position.Volume(),price,0,0));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CloseShort

空头持仓平仓。

```
virtual bool  CloseShort(
   double    price    // 价位
   )
```

参数

*price*

[输入] 价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

空头持仓平仓 (CTrade 类对象的 Buy(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 空头持仓平仓                                                     |
//| 输入:  价位 - 平仓价。                                           |
//| 输出: true-如果交易操作已处理, 否则 false。                      |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CloseShort(double price)
  {
   if(price==EMPTY_VALUE) return(false);
//---
   return(m_trade.Buy(m_position.Volume(),price,0,0));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStop

检查已开仓位的尾随停止条件。

```
virtual bool  CheckTrailingStop()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查已开仓位的尾随停止条件（CheckTrailingStopLong() 或 CheckTrailingStopShort() 分别对应多头或空头仓位）。

实现

```
//+------------------------------------------------------------------+
//| 检查持仓的尾随止损/止盈                                          |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStop()
  {
//--- 持仓必须在调用前被选择
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- 检查修改多头持仓的可能性
      if(CheckTrailingStopLong()) return(true);
     }
   else
     {
      //--- 检查修改空头持仓的可能性
      if(CheckTrailingStopShort()) return(true);
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查已开多头持仓的尾随停止条件。

```
virtual bool  CheckTrailingStopLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查已开多头持仓的尾随停止条件（智能尾随对象的 CheckTrailingStopLong(...) 方法）。如果条件满足，它修改持仓参数（TrailingStopLong(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查多头持仓的尾随止损/止盈                                      |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopLong()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- 检查多头持仓的尾随停止操作
   if(m_trailing.CheckTrailingStopLong(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- 多头持仓的尾随停止操作
      return(TrailingStopLong(sl,tp));
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查已开空头持仓的尾随停止条件。

```
virtual bool  CheckTrailingStopShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查已开空头持仓的尾随停止条件（智能尾随对象的 CheckTrailingStopShort(...) 方法）。如果条件满足，它修改持仓参数（TrailingStopShort(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查空头持仓的尾随止损/止盈                                      |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopShort()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- 检查空头持仓的尾随停止操作
   if(m_trailing.CheckTrailingStopShort(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- 空头持仓的尾随停止操作
      return(TrailingStopShort(sl,tp));
     }
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# TrailingStopLong

它修改已开多头持仓的参数。

```
virtual bool  TrailingStopLong(
   double    sl,    // 止损
   double    tp,    // 止盈
   )
```

参数

*sl*

[输入] 止损价位。

*tp*

[输入] 止损价位。

返回值

true 如果交易操作已被执行，否则 false。

注释

函数修改已开多头持仓的参数 (CTrade 类对象的 PositionModify(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 多头仓位尾随止损/止盈                                            |
//| 输入:  sl - 新的止损,                                           |
//|          tp - 新的止盈。                                         |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopLong(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# TrailingStopShort

它修改已开空头持仓的参数。

```
virtual bool  TrailingStopLong(
   double     sl,     // 止损
   double     tp,     // 止盈
   )
```

参数

*sl*

[输入] 止损价位。

*tp*

[in] Take Profit price.

返回值

true 如果交易操作已被执行，否则 false。

注释

函数修改已开空头持仓的参数 (CTrade 类对象的 PositionModify(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 空头仓位尾随止损/止盈                                            |
//| INPUT:  sl - 新的止损,                                           |
//|         tp - 新的止盈。                                          |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopShort(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingOrderLong

检查低多/高多挂单的尾随停止条件。

```
virtual bool  CheckTrailingOrderLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查低多/高多挂单的尾随停止条件（交易信号对象的 CheckTrailingOrderLong() 方法）并在必要时修改订单参数（TrailingOrderLong(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查低多/高多挂单的尾随条件                                      |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderLong()
  {
   double price;
//--- 检查修改多头订单的可能性
   if(m_signal.CheckTrailingOrderLong(GetPointer(m_order),price))
      return(TrailingOrderLong(m_order.PriceOpen()-price));
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingOrderShort

检查高空/低空挂单的尾随停止条件。

```
virtual bool  CheckTrailingOrderShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查高空/低空挂单的尾随停止条件（交易信号对象的 CheckTrailingOrderShort() 方法）并在必要时修改订单参数（TrailingOrderShort() 方法）。

实现

```
//+------------------------------------------------------------------+
//| 检查高空/低空挂单的尾随                                          |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderShort()
  {
   double price;
//--- 检查修改空头订单的可能性
   if(m_signal.CheckTrailingOrderShort(GetPointer(m_order),price))
      return(TrailingOrderShort(m_order.PriceOpen()-price));
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# TrailingOrderLong

它修改低多/高多挂单的参数。

```
virtual bool  TrailingOrderLong(
   double    delta    // 增量
   )
```

参数

*delta*

[输入] 价格增量。

返回值

true 如果交易操作已被执行，否则 false。

注释

它修改低多/高多挂单的参数(CTrade 类对象的 OrderModify(...)方法)。

实现

```
//+------------------------------------------------------------------+
//| 尾随低多/高多订单                                                |
//| 输入:  delta - 价格变化。                                        |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderLong(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- 修改低多/高多订单
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
  }
```

© 2000-2025, MetaQuotes Ltd.

# TrailingOrderShort

它修改高空/低空挂单的参数。

```
virtual bool  TrailingOrderShort(
   double    delta    // 增量
   )
```

参数

*delta*

[输入] 价格增量。

返回值

true 如果交易操作已被执行，否则 false。

注释

它修改高空/低空挂单的参数 (CTrade 类对象的 OrderModify(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 尾随高空/低空订单                                                |
//| 输入:  delta - 价格变化。                                        |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderShort(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- 修改高空/低空订单
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckDeleteOrderLong

检查删除低多/高多挂单的条件。

```
virtual bool  CheckDeleteOrderLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查挂单的过期时间。它检查删除高空/低空挂单的条件 (信号类对象的 CheckCloseLong(...) 方法) 并在条件满足时删除挂单 (DeleteOrderLong() 方法)。

实现

```
//+------------------------------------------------------------------+
//| 检查删除多头挂单                                                 |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderLong()
  {
   double price;
//--- 检查删除多头订单的可能性
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderLong());
     }
   if(m_signal.CheckCloseLong(price))
      return(DeleteOrderLong());
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# CheckDeleteOrderShort

检查删除高空/低空挂单的条件。

```
virtual bool  CheckDeleteOrderShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它检查挂单的过期时间。它检查删除高空/低空挂单的条件 (信号类对象的 CheckCloseShort(...) 方法) 并在条件满足时删除挂单 (DeleteOrderShort() 方法)。

实现

```
//+------------------------------------------------------------------+
//| 检查删除空头挂单                                                 |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作处理完毕, 否则 false。                    |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderShort()
  {
   double price;
//--- 检查删除空头订单的可能性
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderShort());
     }
   if(m_signal.CheckCloseShort(price))
      return(DeleteOrderShort());
//--- 返回无操作
   return(false);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteOrders

删除所有订单。

```
virtual bool  DeleteOrders()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它删除所有订单 (DeleteOrder() 所有订单)。

实现

```
//+------------------------------------------------------------------+
//| 删除所有订单                                                     |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrders()
  {
   bool result=false;
   int  total=OrdersTotal();
//---
   for(int i=total-1;i>=0;i--)
     {
      if(m_order.Select(OrderGetTicket(i)))
        {
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         result|=DeleteOrder();
        }
     }
//---
   return(result);
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteOrder

删除高空/低空挂单。

```
virtual bool  DeleteOrder()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它删除挂单 (CTrade 类对象的 OrderDelete(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 删除挂单                                                         |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrder()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteOrderLong

删除低多/高多挂单。

```
virtual bool  DeleteOrderLong()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它删除低多/高多挂单 (CTrade 类对象的 OrderDelete(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 删除多头挂单                                                     |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderLong()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

© 2000-2025, MetaQuotes Ltd.

# DeleteOrderShort

删除高空/低空挂单。

```
virtual bool  DeleteOrderShort()
```

返回值

true 如果交易操作已被执行，否则 false。

注释

它删除高空/低空挂单 (CTrade 类对象的 OrderDelete(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 删除空头挂单                                                     |
//| 输入:  无。                                                      |
//| 输出: true-如果交易操作成功, 否则 false。                        |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderShort()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

© 2000-2025, MetaQuotes Ltd.

# LotOpenLong

获取多头操作交易量。

```
double  LotOpenLong(
   double    price,   // 价位
   double    sl       // 止损
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头操作（手数）交易量。

注释

它获取多头操作交易量（资金管理对象的 CheckOpenLong(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 获取开多头仓位手数的方法。                                       |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损。                                          |
//| 输出: 开仓手数。                                                 |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
double CExpert::LotOpenLong(double price,double sl)
  {
   return(m_money.CheckOpenLong(price,sl));
  }
```

© 2000-2025, MetaQuotes Ltd.

# LotOpenShort

获取空头操作交易量。

```
double  LotOpenShort(
   double    price,   // 价位
   double    sl       // 止损
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

空头操作（手数）交易量。

注释

它获取空头操作交易量（资金管理对象的 CheckOpenShort(...) 方法）。

实现

```
//+------------------------------------------------------------------+
//| 获取开空头仓位手数的方法。                                       |
//| 输入:  price - 价位,                                             |
//|          sl    - 止损。                                          |
//| 输出: 开仓手数。                                                 |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
double CExpert::LotOpenShort(double price,double sl)
  {
   return(m_money.CheckOpenShort(price,sl));
  }
```

© 2000-2025, MetaQuotes Ltd.

# LotReverse

获取仓位反转操作交易量。

```
double  LotReverse(
   double    sl       // 止损
   )
```

参数

*sl*

[输入] 止损价位。

返回值

仓位反转操作交易量 (手数)。

注释

它获取仓位反转操作交易量 (资金管理对象的 CheckReverse(...) 方法)。

实现

```
//+------------------------------------------------------------------+
//| 获取仓位反转手数的方法。                                         |
//| 输入:  sl - 止损。                                               |
//| 输出: 开仓手数。                                                 |
//| 备注: 无。                                                       |
//+------------------------------------------------------------------+
double CExpert::LotReverse(double sl)
  {
   return(m_money.CheckReverse(GetPointer(m_position),sl));
  }
```

© 2000-2025, MetaQuotes Ltd.

# PrepareHistoryDate

设置交易历史跟踪的开始日期。

```
void  PrepareHistoryDate()
```

注释

交易历史跟踪周期的设置自月初开始 (但是不能少于一天)。

© 2000-2025, MetaQuotes Ltd.

# HistoryPoint

创建交易历史的检测点 (保存仓位，订单，成交和历史订单的数量)。

```
void  HistoryPoint(
   bool   from_check_trade=false   // 标志
   )
```

参数

*from_check_trade=false*

[输入] 避免递归的标志。

注释

它保存仓位，订单，成交和历史订单的数量。

© 2000-2025, MetaQuotes Ltd.

# CheckTradeState

当前状态和之前的保存记录进行比较，并调用相应的事件处理器。

```
bool  CheckTradeState()
```

返回值

true 如果事件已经被处理，否则 false。

注释

它检查当前仓位，订单，成交和历史订单的数量，并与之前通过 HistoryPoint() 方法保存的记录进行比较。如果交易历史有变化，它调用相应的虚事件处理器。

© 2000-2025, MetaQuotes Ltd.

# WaitEvent

设置事件等待标志。

```
void  WaitEvent(
   ENUM_TRADE_EVENTS    event          // 标志
   )
```

参数

*event*

[输入] 设置的事件标志 (ENUM_TRADE_EVENTS 枚举)。

返回值

无。

事件标志

```
//--- 期望事件的标志
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT                =0,         // 无期望事件
   TRADE_EVENT_POSITION_OPEN           =0x1,       // 期望的 "开仓" 事件标志
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,         // 期望的 "修改交易量" 事件标志
   TRADE_EVENT_POSITION_MODIFY         =0x4,       // 期望的 "修改持仓的停止订单" 事件标志
   TRADE_EVENT_POSITION_CLOSE          =0x8,       // 期望的 "平仓" 事件标志
   TRADE_EVENT_POSITION_STOP_TAKE      =0x10,      // 期望的 "触发仓位停止订单" 事件标志
   TRADE_EVENT_ORDER_PLACE             =0x20,      // 期望的 "放置挂单" 事件标志
   TRADE_EVENT_ORDER_MODIFY            =0x40,      // 期望的 "修改挂单" 事件标志
   TRADE_EVENT_ORDER_DELETE            =0x80,      // 期望的 "删除挂单" 事件标志
   TRADE_EVENT_ORDER_TRIGGER           =0x100      // 期望的 "触发挂单" 事件标志
  };
```

© 2000-2025, MetaQuotes Ltd.

# NoWaitEvent

重置事件等待标志。

```
void  NoWaitEvent(
   ENUM_TRADE_EVENTS    event          // 标志
   )
```

参数

*event*

[输入] 事件重置的标志 (ENUM_TRADE_EVENTS 枚举)。

返回值

无。

事件标志

```
//--- 期望事件的标志
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT                =0,         // 无期望事件
   TRADE_EVENT_POSITION_OPEN           =0x1,       // 期望的 "开仓" 事件标志
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,         // 期望的 "修改交易量" 事件标志
   TRADE_EVENT_POSITION_MODIFY         =0x4,       // 期望的 "修改持仓的停止订单" 事件标志
   TRADE_EVENT_POSITION_CLOSE          =0x8,       // 期望的 "平仓" 事件标志
   TRADE_EVENT_POSITION_STOP_TAKE      =0x10,      // 期望的 "触发仓位停止订单" 事件标志
   TRADE_EVENT_ORDER_PLACE             =0x20,      // 期望的 "放置挂单" 事件标志
   TRADE_EVENT_ORDER_MODIFY            =0x40,      // 期望的 "修改挂单" 事件标志
   TRADE_EVENT_ORDER_DELETE            =0x80,      // 期望的 "删除挂单" 事件标志
   TRADE_EVENT_ORDER_TRIGGER           =0x100      // 期望的 "触发挂单" 事件标志
  };
```

© 2000-2025, MetaQuotes Ltd.

# TradeEventPositionStopTake

"仓位止损/止盈触发"事件处理器。

```
virtual bool  TradeEventPositionStopTake()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventOrderTriggered

"挂单触发"事件处理器。

```
virtual bool  TradeEventOrderTriggered()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventPositionOpened

"开仓触发"事件处理器。

```
virtual bool  TradeEventPositionOpened()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventPositionVolumeChanged

"持仓量改变"事件处理器。

```
virtual bool  TradeEventPositionVolumeChanged()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventPositionModified

"持仓修改"事件处理器。

```
virtual bool  TradeEventPositionModified()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventPositionClosed

"平仓"事件处理器。

```
virtual bool  TradeEventPositionClosed()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventOrderPlaced

"放置挂单"事件处理器。

```
virtual bool  TradeEventOrderPlaced()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventOrderModified

"挂单修改"事件处理器。

```
virtual bool  TradeEventOrderModified()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventOrderDeleted

"挂单删除"事件处理器。

```
virtual bool  TradeEventOrderDeleted()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# TradeEventNotIdentified

无标识事件处理器。

```
virtual bool  TradeEventNotIdentified()
```

返回值

类 CExpert 方法不做任何事且永远返回 true。

注释

需要注意的是有些交易事件可以抵达，在这种情况下，很难辨别它们。

© 2000-2025, MetaQuotes Ltd.

# TimeframeAdd

添加时间帧用于跟踪。

```
void  TimeframeAdd(
   ENUM_TIMEFRAMES    period          // 时间帧
   )
```

参数

*period*

[输入] 时间帧 (ENUM_TIMEFRAMES 枚举)。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# TimeframesFlags

方法返回新柱线的时间帧标志指示。

```
int  TimeframesFlags(
   MqlDateTime&     time          // 时间变量
   )
```

参数

*time*

[输入] 新时间的 MqlDateTime 类型变量，通过引用传递。

返回值

返回的标志，指示新柱线的时间帧。

© 2000-2025, MetaQuotes Ltd.

# CExpertSignal

CExpertSignal 是交易信号的基准类，它不做任何操作（除了 CheckReverseLong() 和 CheckReverseShort() 方法）但提供了接口。

如何使用它：

1. 准备交易信号的算法；
2. 创建您自己的，继承自 CExpertSignal的交易信号类；
3. 在您的类中用您自己的算法覆盖虚方法。

您可以在 Expert\Signal\ 文件夹里找到交易信号类的例程。

## 描述

CExpertSignal是实现交易信号算法的基准类。

## 声明

```
class CExpertSignal : public CExpertBase
```

## 标称库文件

```
#include <Expert\ExpertSignal.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertSignal

直接分支

CSignalAC, CSignalAMA, CSignalAO, CSignalBearsPower, CSignalBullsPower, CSignalCCI, CSignalDeM, CSignalDEMA, CSignalEnvelopes, CSignalFrAMA, CSignalRSI, CSignalRVI, CSignalSAR, CSignalStoch, CSignalTEMA, CSignalTriX, CSignalWPR

## 类方法

| 初始化 | |
|---|---|
| virtual InitIndicators | 初始化指标和时间序列 |
| virtual ValidationSettings | 验证设置 |
| virtual AddFilter | 在组合信号里添加过滤器 |
| 访问保护数据 | |
| BasePrice | 设置基准价位 |
| UsedSeries | 获取时间序列使用的位掩码。 |
| 参数设置 | |

© 2000-2025, MetaQuotes Ltd.

| 初始化 | |
|---|---|
| 权值 | 设置"权重"参数的值 |
| PatternsUsage | 设置"形态用法"参数的值 |
| General | 设置"常用"参数的值 |
| Ignore | 设置"忽略"参数的值 |
| Invert | 设置"反向"参数的值 |
| ThresholdOpen | 设置"开仓阀值"参数的值 |
| ThresholdClose | 设置"平仓阀值"参数的值 |
| PriceLevel | 设置"价位"参数的值 |
| StopLevel | 设置"止损位"参数的值 |
| TakeLevel | 设置"止盈位"参数的值 |
| Expiration | 设置"过期"参数的值 |
| Magic | 设置"魔幻数字"参数的值 |
| 检查交易条件 | |
| virtual CheckOpenLong | 检查开多头仓位的条件 |
| virtual CheckCloseLong | 检查平多头仓位的条件 |
| virtual CheckOpenShort | 检查开空头仓位的条件 |
| virtual CheckCloseShort | 检查平空头仓位的条件 |
| virtual CheckReverseLong | 检查多头仓位反转条件 |
| virtual CheckReverseShort | 检查空头仓位反转条件 |
| 交易参数设置 | |
| virtual OpenLongParams | 设置开多头仓位的参数 |
| virtual OpenShortParams | 设置开空头仓位的参数 |
| virtual CloseLongParams | 设置多头仓位平仓的参数 |
| virtual CloseShortParams | 设置空头仓位平仓的参数 |
| 检查订单尾随条件 | |
| virtual CheckTrailingOrderLong | 检查修改多头挂单参数的条件 |
| virtual CheckTrailingOrderShort | 检查修改空头挂单参数的条件 |
| 检查市价订单形成的方法 | |
| virtual LongCondition | 获取多头条件检查的结果 |

© 2000-2025, MetaQuotes Ltd.

| 初始化 | |
|---|---|
| virtual ShortCondition | 获取空头条件检查的结果 |
| virtual Direction | 获取价格的"加权"方向 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

© 2000-2025, MetaQuotes Ltd.

# BasePrice

设置基准价位。

```
void  BasePrice(
   double    value          // 新值
   )
```

参数

*value*

[输入] 基准价位的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# UsedSeries

获取时间序列使用的标志。

```
int  UsedSeries()
```

返回值

时间序列的使用标志 (如果品名/时间帧与工作时的品名/时间帧相应), 否则 0。

© 2000-2025, MetaQuotes Ltd.

# 权值

设置 "加权" 参数的新值。

```
void  Weight(
   double    value          // 新值
   )
```

参数

*value*

[输入] "加权" 的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# PatternUsage

设置"形态用法"参数的新值。

```
void  PatternUsage(
   double    value          // 新值
   )
```

参数

*value*

[输入]"形态用法"参数的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# General

设置"常用"参数的值。

```
void  General(
   int     value          // 新值
   )
```

参数

*value*

[输入]"常用"的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Ignore

设置"忽略"参数的值。

```
void  Ignore(
   long     value          // 新值
   )
```

参数

*value*

[输入]"忽略"的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Invert

设置"反向"参数的值。

```
void  Invert(
   long     value          // 新值
   )
```

参数

*value*

[输入] 新"反向"值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ThresholdOpen

设置"开仓阀值"参数的新值。

```
void  ThresholdOpen(
   long     value          // 新值
   )
```

参数

*value*

[输入]"开仓阀值"的新值。

返回值

无。

注释

"开仓阀值"参数的范围从 0 至 100. 用于为开仓"投票"。

© 2000-2025, MetaQuotes Ltd.

# ThresholdClose

设置"平仓阀值"参数的新值。

```
void  ThresholdOpen(
   long     value          // 新值
   )
```

参数

*value*

[输入]"平仓阀值"的新值。

返回值

无。

注释

"平仓阀值"参数的范围从 0 至 100. 用于为平仓"投票"。

© 2000-2025, MetaQuotes Ltd.

# PriceLevel

设置"价位"参数的新值。

```
void  PriceLevel(
   double    value          // 新值
   )
```

参数

*value*

[输入]"价位"的新值。

返回值

无。

注释

"价位"值以价格单位定义。价格单位的数字由 PriceLevelUnit() 方法返回。"价位"用于定义相对基准价的开仓价。

© 2000-2025, MetaQuotes Ltd.

# StopLevel

设置"止损位"参数的新值。

```
void  StopLevel(
   double    value          // 新值
   )
```

参数

*value*

[输入]"止损位"的新值。

返回值

无。

注释

"止损位"的值以价格单位定义。价格单位的数字由 PriceLevelUnit() 方法返回。"止损位"用于定义相对基准价的止损位。

© 2000-2025, MetaQuotes Ltd.

# TakeLevel

设置"止盈位"参数的新值。

```
void  TakeLevel(
   double    value          // 新值
   )
```

参数

*value*

[输入]"止盈位"的新值。

返回值

无。

注释

"止盈位"的值以价格单位定义。价格单位的数字由 PriceLevelUnit() 方法返回。"止盈位"用于定义相对基准价的止盈位。

© 2000-2025, MetaQuotes Ltd.

# Expiration

设置"过期"参数的值。

```
void  Expiration(
   int     value          // 新值
   )
```

参数

*value*

[输入]"过期"的新值。

返回值

无。

注释

"过期"参数的值以柱线为单位定义。它作为挂单的过期时间（当交易采用挂单时）。

© 2000-2025, MetaQuotes Ltd.

# Magic

设置"魔幻数字"参数的值。

```
void  Magic(
   int     value          // 新值
   )
```

参数

*value*

[输入]"魔幻数字"(智能交易 ID)的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# InitIndicators

初始化所有指标和时间序列。

```
virtual bool  InitIndicators(
   CIndicators*  indicators    // 指针
   )
```

参数

*indicators*

[输入] 指标和时间序列集合的指针。

返回值

true 如果成功，否则 false。

注释

如果对象使用了品名或时间帧则时间序列被初始化，与初始化时定义的品名或时间帧不同。

© 2000-2025, MetaQuotes Ltd.

# AddFilter

在复合信号里添加过滤器。

```
virtual bool  AddFilter(
   CExpertSignal*  filter    // 指针
   )
```

参数

*indicators*

[输入] 过滤器对象的指针。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

检查开多头仓位的条件。

```
virtual bool  CheckOpenLong(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

检查开空头仓位的条件。

```
virtual bool  CheckOpenShort(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OpenLongParams

设置多头仓位开仓的参数。

```
virtual bool  OpenLongParams(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OpenShortParams

设置空头仓位开仓的参数。

```
virtual bool  OpenShortParams(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckCloseLong

检查平多头仓位的条件。

```
virtual bool  CheckCloseLong(
   double&  price      // 价位
   )
```

参数

*price*

[输入][输出] 平仓价位变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckCloseShort

检查平空头仓位的条件。

```
virtual bool  CheckCloseShort(
   double&  price      // 价位
   )
```

参数

*price*

[输入][输出] 平仓价位变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CloseLongParams

设置多头仓位平仓的参数。

```
virtual bool  CloseLongParams(
   double&    price          // 价位
   )
```

参数

*price*

[输入][输出] 平仓价位变量，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CloseShortParams

设置空头仓位平仓的参数。

```
virtual bool  CloseShortParams(
   double&    price          // 价位
   )
```

参数

*price*

[输入][输出] 平仓价位变量，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckReverseLong

检查多头仓位反转条件。

```
virtual bool  CheckReverseLong(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckReverseShort

检查空头仓位反转条件。

```
virtual bool  CheckReverseShort(
   double&    price,          // 价位
   double&    sl,             // 止损
   double&    tp,             // 止盈
   datetime&  expiration      // 过期
   )
```

参数

*price*

[输入][输出] 反转价位变量，通过引用传递。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

*expiration*

[输入][输出] 过期时间变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingOrderLong

检查修改多头挂单参数的条件。

```
virtual bool  CheckTrailingOrderLong(
   COrderInfo*     order,          // 订单
   double&         price           // 价位
   )
```

参数

*order*

[输入] 类COrderInfo对象的指针。

*price*

[输入][输出] 止损价变量。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingOrderShort

检查修改空头挂单参数的条件。

```
virtual bool  CheckTrailingOrderShort(
   COrderInfo*     order,          // 订单
   double&         price           // 价位
   )
```

参数

*order*

[输入] 类COrderInfo对象的指针。

*price*

[输入][输出] 止损价变量。

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# LongCondition

检查开多头仓位的条件。

```
virtual int  LongCondition()
```

返回值

当条件满足时，它返回从 1 至 100 之间的值（依据信号 "强度"），如果没有开多头仓位的信号，它返回 0。

注释

基准类的 LongCondition() 方法并未实现多头开仓条件检查且永远返回 0。

© 2000-2025, MetaQuotes Ltd.

# ShortCondition

检查开空头仓位的条件。

```
virtual int  ShortCondition()
```

返回值

当条件满足时，它返回从 1 至 100 之间的值（依据信号 "强度"），如果没有开空头仓位的信号，它返回 0。

注释

基准类的 ShortCondition() 方法并未实现空头开仓条件检查且永远返回 0。

© 2000-2025, MetaQuotes Ltd.

# Direction

返回"加权"方向的值。

```
virtual double  Direction()
```

返回值

当上行方向时返回值>0（可能），且当下行方向时返回值<0。绝对值则依据信号的"强度"。

注释

如果使用过滤器，则结果将依据过滤器。

© 2000-2025, MetaQuotes Ltd.

# CExpertTrailing

CExpertTrailing 是尾随算法的基准类，它不做任何操作，但提供接口。

如何使用它：

1. 准备尾随算法；
2. 创建您自己的继承 CExpertTrailing 的子类；
3. 在您的类中用您自己的算法覆盖虚方法。

您可以在 Expert\Trailing\ 文件夹里找到尾随类的例程。

## 描述

CExpertTrailing 是实现尾随停止算法的基准类。

## 声明

```
class CExpertTrailing : public CExpertBase
```

## 标称库文件

```
#include <Expert\ExpertTrailing.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertTrailing

直接分支

CTrailingFixedPips, CTrailingMA, CTrailingNone, CTrailingPSAR

## 类方法

| 检查尾随停止条件 | |
|---|---|
| virtual CheckTrailingStopLong | 检查修改多头仓位参数的条件 |
| virtual CheckTrailingStopShort | 检查修改空头仓位参数的条件 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries, InitIndicators

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查修改多头仓位参数的条件。

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // 指针
   double&         sl,           // 止损
   double&         tp            // 止盈
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

注释

基准类的 CheckTrailingStopLong(...) 方法永远返回 false。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查修改空头仓位参数的条件。

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // 指针
   double&         sl,           // 止损
   double&         tp            // 止盈
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量，通过引用传递。

*tp*

[输入][输出] 止盈价变量，通过引用传递。

返回值

true 如果条件满足，否则 false。

注释

基准类的 CheckTrailingStopShort(...) 方法永远返回 false。

© 2000-2025, MetaQuotes Ltd.

# CExpertMoney

CExpertMoney 是资金和风险管理算法的基准类。

## 描述

CExpertMoney 是实现资金和风险管理的基准类。

## 声明

```
class CExpertMoney : public CObject
```

## 标称库文件

```
#include <Expert\ExpertMoney.mqh>
```

## 继承体系

CObject

CExpertBase

CExpertMoney

直接分支

CMoneyFixedLot, CMoneyFixedMargin, CMoneyFixedRisk, CMoneyNone, CMoneySizeOptimized

## 类方法

| 访问保护数据 | |
|---|---|
| Percent | 设置"风险百分比"参数的值 |
| 初始化 | |
| virtual ValidationSettings | 验证设置 |
| 交易条件检查 | |
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |
| virtual CheckReverse | 获取反转仓位的交易量 |
| virtual CheckClose | 检查已开仓位平仓条件 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

© 2000-2025, MetaQuotes Ltd.

# Percent

设置"风险百分比"参数的值。

```
void  Percent(
   double  percent      // 风险百分比
   )
```

参数

*percent*

[输入] 风险百分比。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

基准类的 ValidationSettings() 方法永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价格
   double  sl         // 止损
   )
```

参数

*price*

[输入] 多头仓位的开仓价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取空头仓位的交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价格
   double  sl         // 止损
   )
```

参数

*price*

[输入] 空头仓位的开仓价位。

*sl*

[输入] 止损价位。

返回值

空头仓位的交易量。

© 2000-2025, MetaQuotes Ltd.

# CheckReverse

获取反转仓位的交易量。

```
virtual double  CheckReverse(
   CPositionInfo*  position,     // 指针
   double          sl            // 止损
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入] 止损价位。

返回值

反转仓位的交易量。

© 2000-2025, MetaQuotes Ltd.

# CheckClose

检查已开仓位平仓条件.

```
virtual double  CheckClose()
```

返回值

true 如果条件满足，否则 false。

© 2000-2025, MetaQuotes Ltd.

# 交易信号模块

在标准交付的客户终端里包括一套现成的可用于 "MQL5 向导" 的交易信号模块。当在 MQL5 向导里创建智能交易时，您可以使用任意的交易信号模块组合 (高达 64 个)。最终的交易操作决策，将通过复杂分析所有包括模块中获取的信号来制订。有关制订交易决策机制的详述在 以下 给定。

在标准交付里包括以下信号模块。:

- 加速计振荡器指标信号
- 自适应移动均线指标信号
- 动量振荡器指标信号
- 空头力度振荡器信号
- 多头力度振荡器信号
- 商品通道指数振荡器信号
- DeMarker振荡器信号
- 双重指数移动均线指标信号
- 包络线指标信号
- 分形自适应移动均线指标信号
- 日内时间过滤信号
- MACD 振荡器信号
- 移动均线指标信号
- 抛物线转向指标信号
- 相对强度指数振荡器信号
- 相对动量指数振荡器信号
- 随机振荡器信号
- 三重指数均线振荡器信号
- 三重指数移动均线指标信号
- 威廉姆斯百分比范围振荡器信号

## 基于模块信号基础上的制订交易决策机制

制订交易决策机制可以呈现为以下基本原理的列表:

- 每个信号模块均有自己的一套市场模型 (确定的价格和指标值组合)。
- 每套市场模型均有显著的变化范围从 1 至 100。越高越明显，模型越健壮。
- 每套模型都可生成价格走势方向的预测。
- 一个模块的预测是搜索内嵌模型的结果，且其输出是 -100 到 100 范围内的数字。该符号确定预测走势的方向 (负符号意味着价格将要下跌，正符号意味着价格将要上涨)。绝对值对应于所发现的最佳模型的强度。
- 每个模块的预测可向最终的 "投票" 发送加权系数 0 至 1，这个值在设置中的 ("Weight") 里指定。
- 投票结果是一个范围在-100 至 100 的数字，其符号确定预测走势的方向，且绝对值象征信号强度。它通过计算所有信号模块的加权预测值的算术平均值得来。

© 2000-2025, MetaQuotes Ltd.

每款生成的智能交易有两个可调整设置 — 开仓和平仓阀值 (ThresholdOpen 和 ThresholdClose) 其值为 0 至 100 范围内的数值。如果最终的信号强度超出阀值级别, 则对应于信号符号的交易操作被执行。

## 举例

考虑一款智能交易带有以下阀值:ThresholdOpen=20 和 ThresholdClose=90。两个信号模块参与制订交易操作决策:移动均线 模块加权值 0.4 以及 随机振荡随机振荡 模块加权值 0.8。让我们来分析两种得自交易信号的变化:

变种 1.

价格上穿上升的移动均线。这种情况与在 MA 模块 里实现的市场模型相对应。此模型意味着价格上涨。它的重要性等于100。与此同时, 随机振荡器返身向下且形成价格背离。这种情况与在 随机振荡模块 里实现的模型相对应。此模型意味着价格下跌。此模型的权值是 80。

让我们来计算最终的 "投票" 结果。计算从 MA 模块获得的比率为 0.4 * 100 = 40。计算从随机振荡模块获得的比率为 0.8 * (-80) = -64。计算两个比率的算术平均值得到最终结果: (40 - 64)/2 = -12。投票结果是做空信号, 相对强度等于 12。而阀值级别等于 20, 尚未触及。所以交易操作没有被执行。

变种 2.

价格下穿上升的移动均线。这种情况与在 MA 模块 里实现的模型相对应。此模型意味着价格的上涨。它的重要性等于10。与此同时, 随机振荡器返身向下且形成价格背离。这种情况与在 随机振荡模块 里实现的模型相对应。此模型意味着价格下跌。此模型的权值是 80。

让我们来计算最终的 "投票" 结果。计算从 MA 模块获得的比率为 0.4 * 10 = 4. 计算从随机振荡模块获得的比率为 0.8 * (-80) = -64。计算两个比率的算术平均值得到最终结果: (4 - 64)/2 = -30。投票结果是做空信号, 相对强度等于 30。而阀值级别等于 20 已触及。所以信号结果是开空头仓位。

a) 价格和随机振荡器背离 (变种 1 和 2)。

© 2000-2025, MetaQuotes Ltd.

b) 价格上穿 MA 指标 (变种 1)。

c) 价格下穿 MA 指标 (变种 2)。

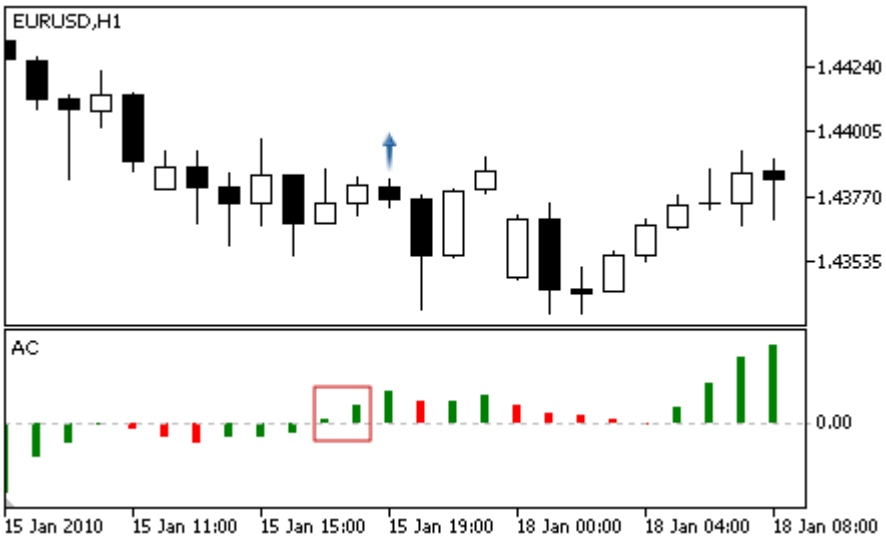
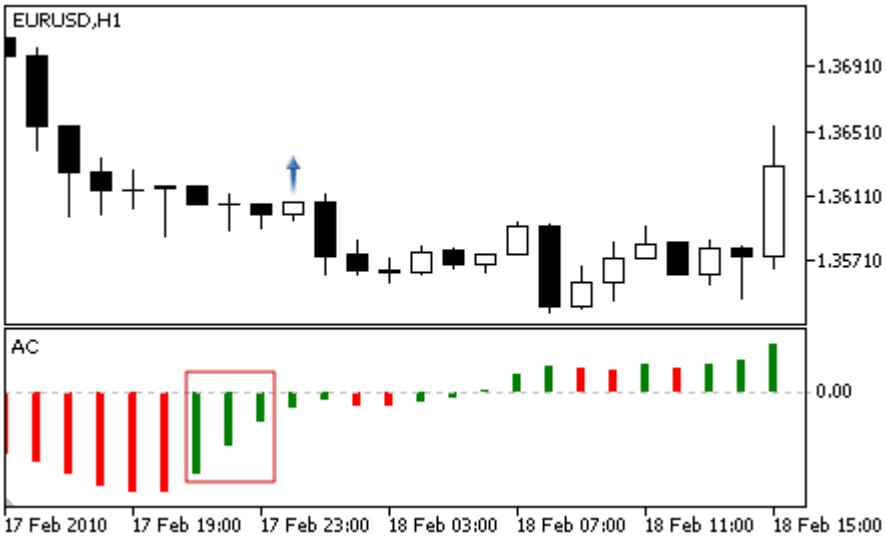
© 2000-2025, MetaQuotes Ltd.

# 加速计振荡器指标信号

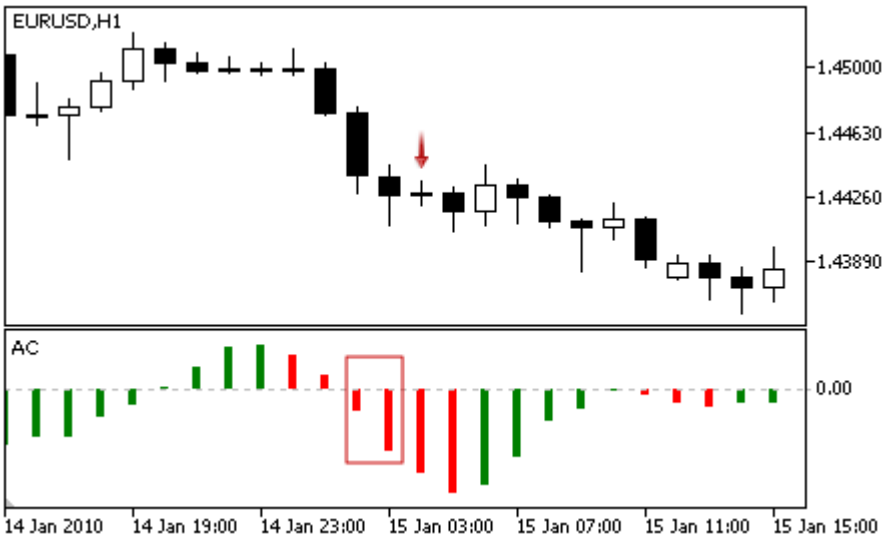
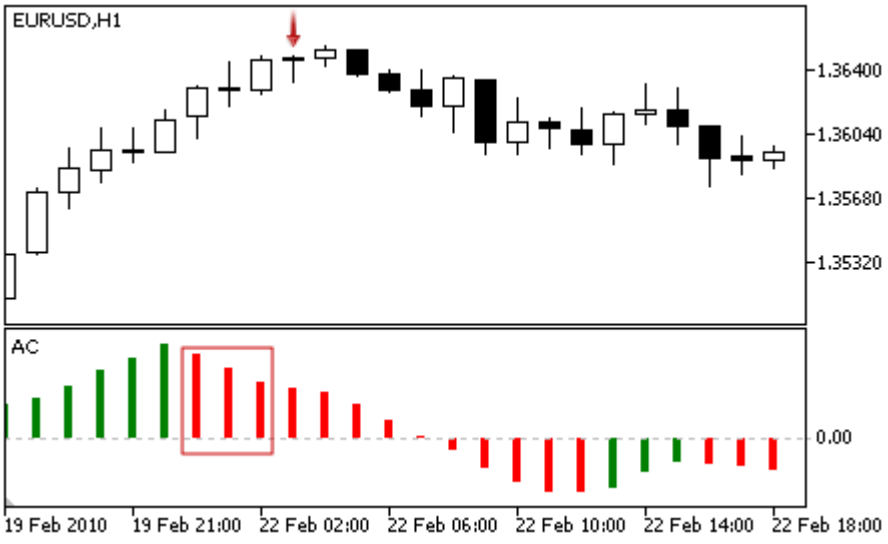
该信号模块基于 加速计振荡器 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 指标值高于 0，且在所分析的柱线中继续上涨，高于前根柱线。<br>  <br>• 指标值低于 0，且在所分析的柱线中继续上涨，高于前根柱线。<br>  |
| 对于做空 | • 指标值低于 0，且在所分析的柱线中继续下跌，低于前根柱线。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 指标值低于 0，且在所分析的柱线中继续下跌，低于前根柱线。<br>  |
| 不反对做多 | 指标值在所分析的柱线中上涨。 |
| 不反对做空 | 指标值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

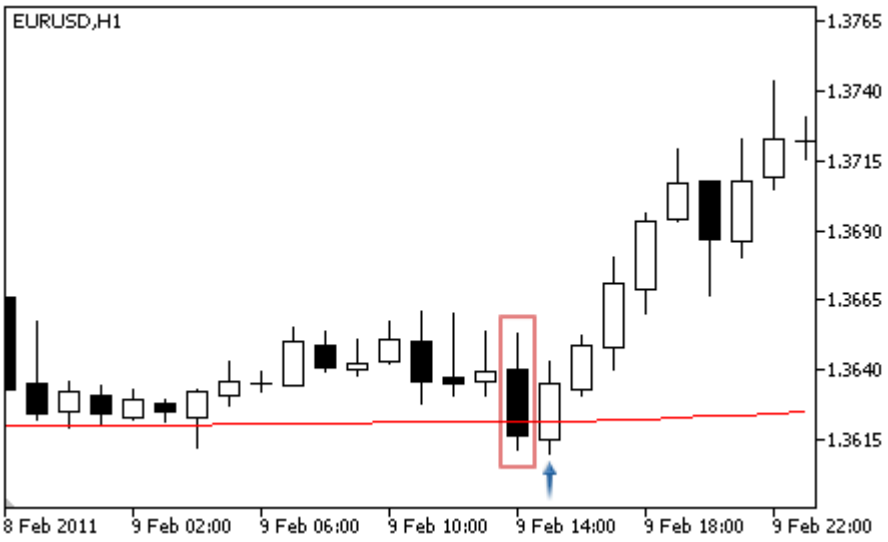
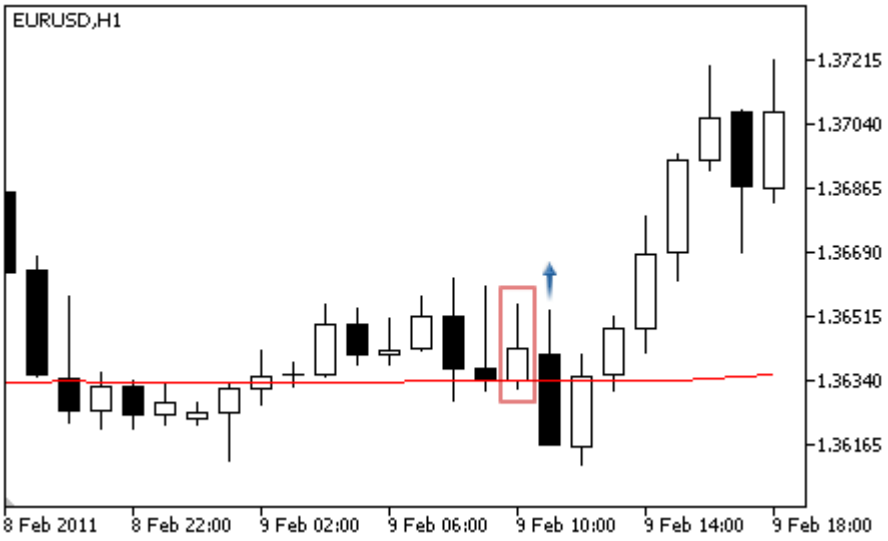
© 2000-2025, MetaQuotes Ltd.

# 自适应移动均线指标信号

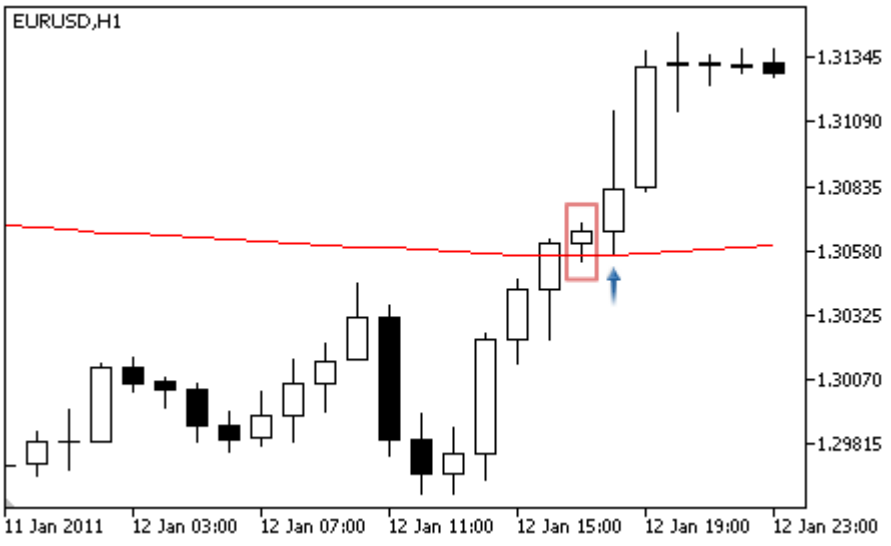
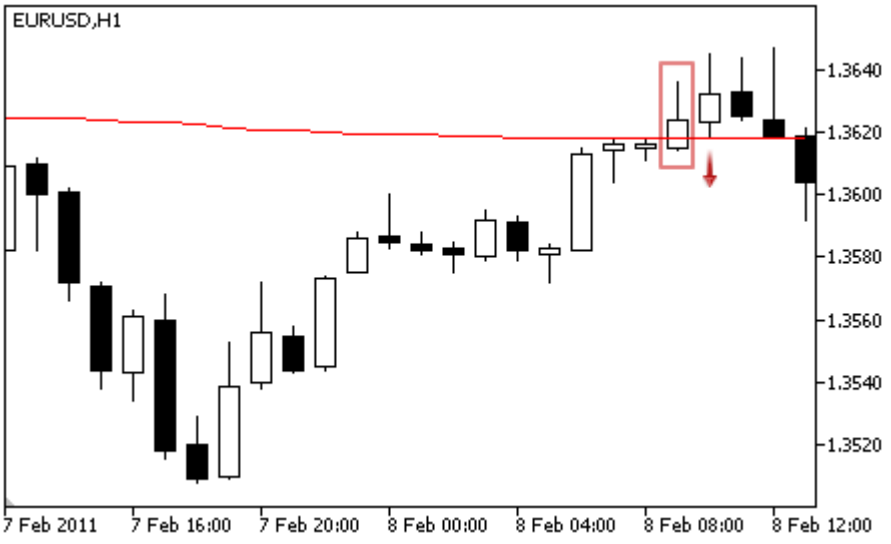
该信号模块基于 自适应移动均线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

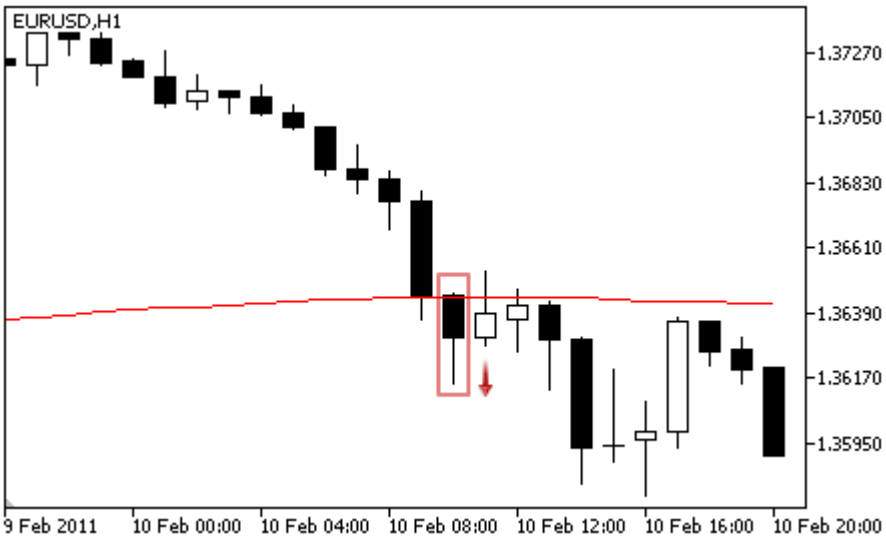
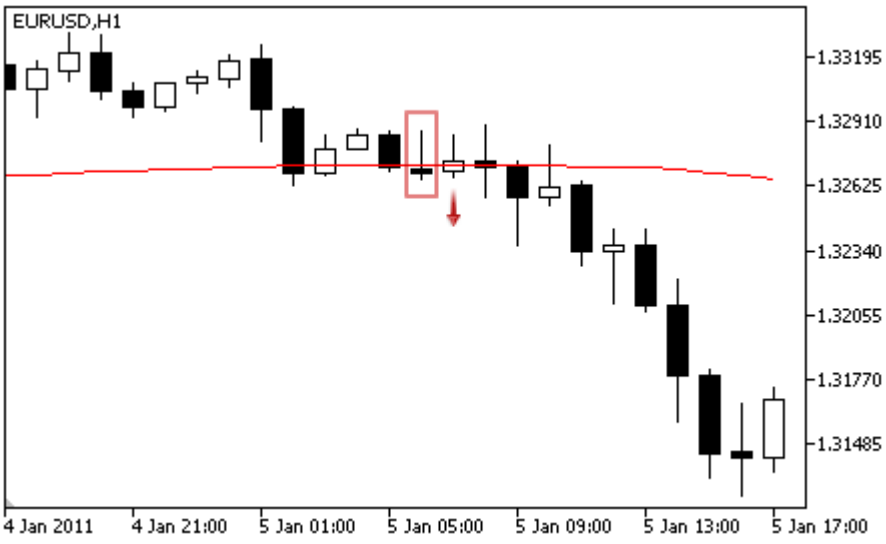
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标上涨（弱信号）。<br>  <br><br>• **移动均线穿越**。价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标上涨（强信号）。<br>  <br><br>• 柱线的下影线穿越指标（所分析柱线的开盘价和收盘价均高于指标，而最低价低于指标）且指标上涨（弱信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标下跌（弱信号）。<br>  <br>• **移动均线穿越**。价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标下跌（强信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 柱线的上影线穿越指标（所分析柱线的开盘价和收盘价均低于指标，而最高价高于指标）且指标下跌（弱信号）。<br>  |
| 不反对做多 | 价格在指标之上。 |
| 不反对做空 | 价格在指标之下。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 均线指标的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |

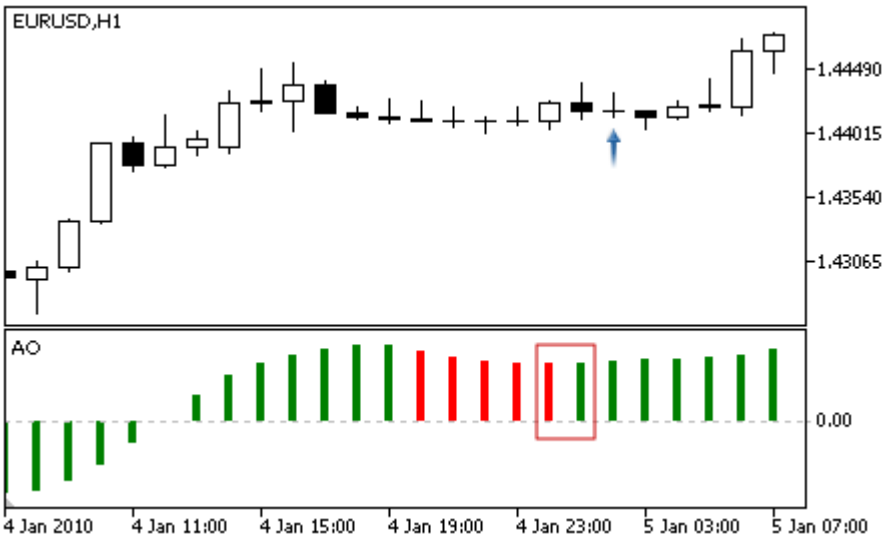
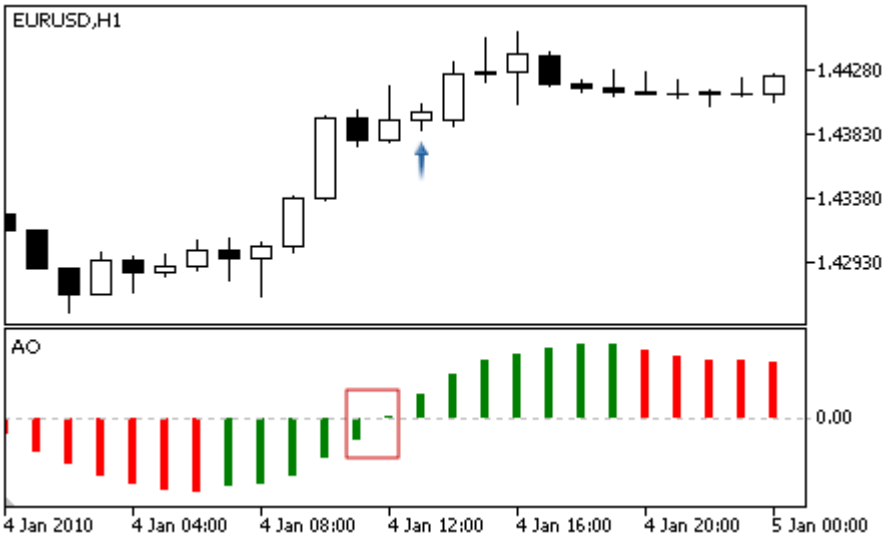
© 2000-2025, MetaQuotes Ltd.

# 动量振荡器指标信号

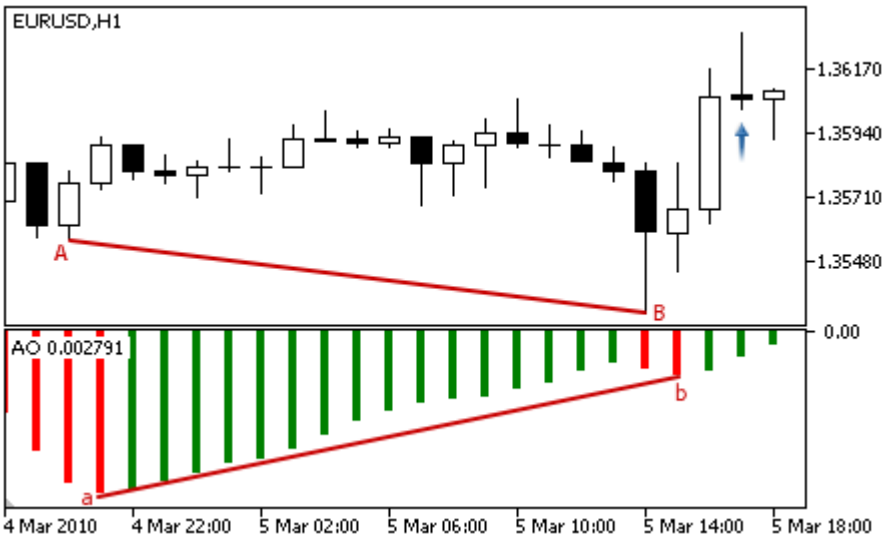
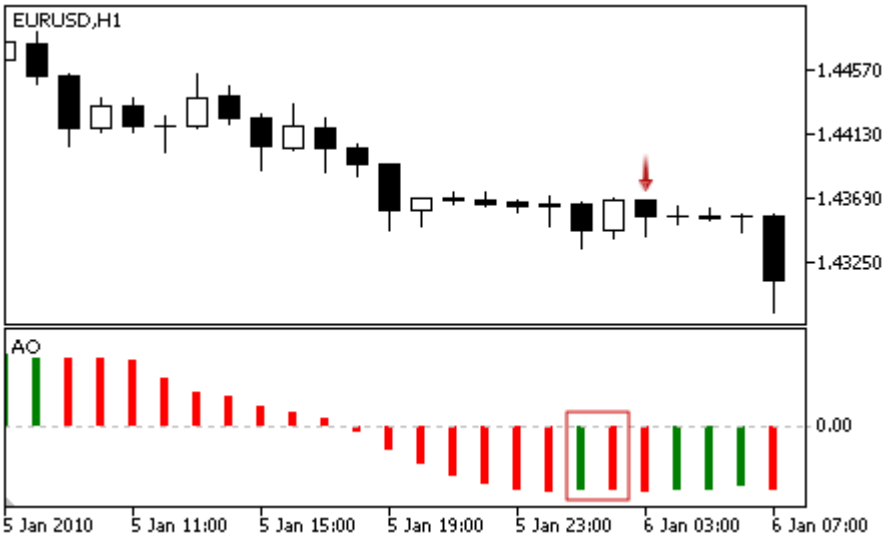
该信号模块基于 动量振荡器 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

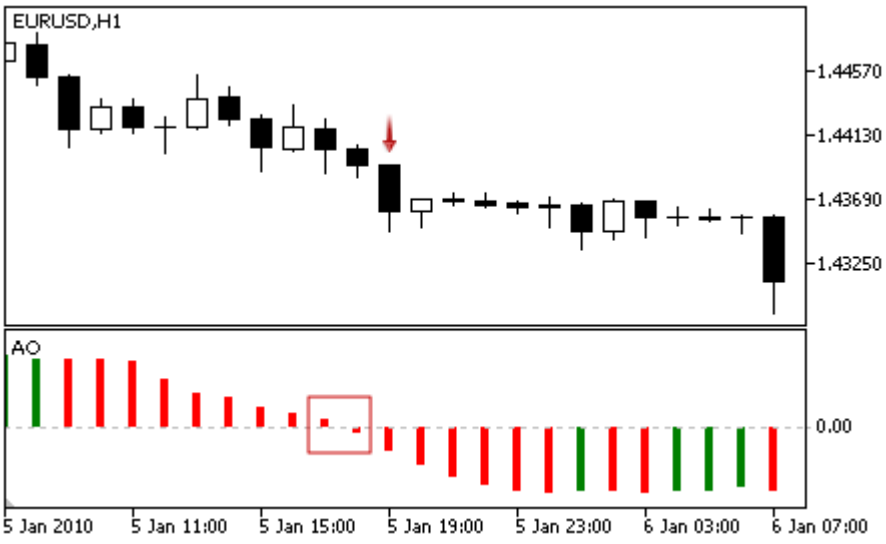
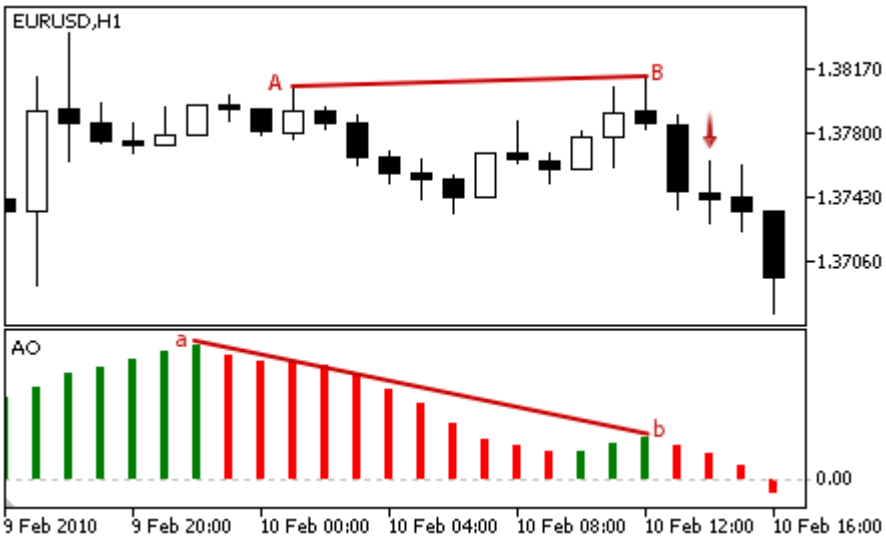
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 蝶形 —指标值在所分析的柱线中上涨，但在前根柱线时下跌，且在两根柱线上的值均高于 0。<br>  <br>• 穿越零轴 — 指标值在所分析的柱线上高于 0，且在前根柱线上低于 0。<br>  <br>• 背离 — 指标最近分析的底部比更早的一个要浅，而相应的价格波谷比更早的一个要深。此外，指标必须不能涨过零轴。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 蝶形 —指标值在所分析的柱线中下跌，但在前根柱线时上涨，且在两根柱线上的值均低于 0。<br>  <br>• 穿越零轴 —指标值在所分析的柱线上低于 0，且在前根柱线上高于 0。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 背离 — 指标最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。此外，指标必须不能跌过零轴。<br>  |
| 不反对做多 | 指标值在所分析的柱线中上涨。 |
| 不反对做空 | 指标值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

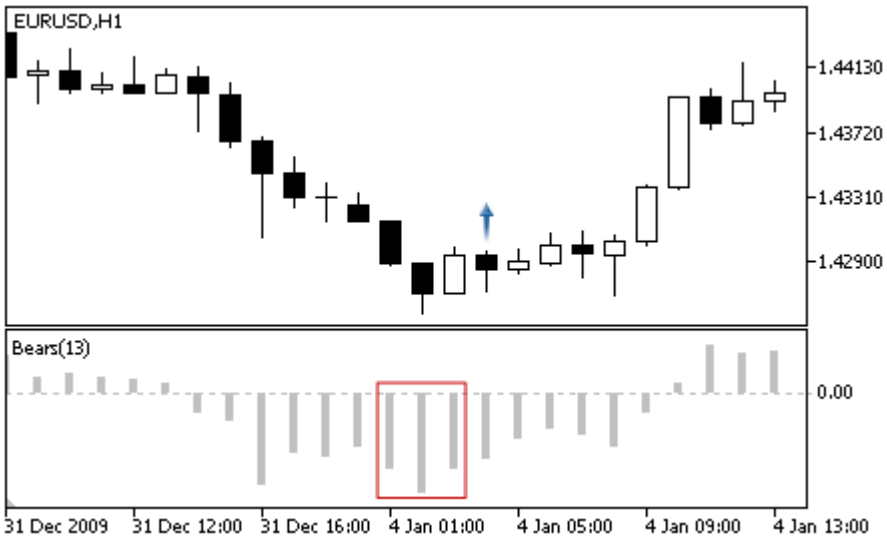
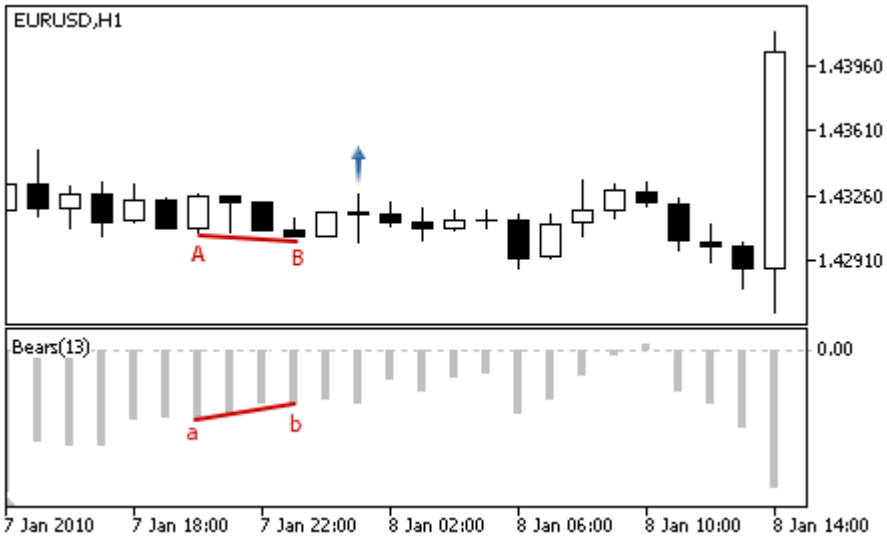
© 2000-2025, MetaQuotes Ltd.

# 空头力度振荡器信号

该信号模块基于 空头力度 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 反转 —振荡器返身向上，而它在所分析的柱线上的值低于 0。<br>  <br>• 背离 —振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。此外，振荡器必须不能涨过零轴。<br>  |
| 对于做空 | 无做空信号。 |
| 不反对做多 | 振荡器的值小于 0。 |
| 不反对做空 | 无信号。 |

© 2000-2025, MetaQuotes Ltd.

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| PeriodBears | 振荡器计算的周期。 |

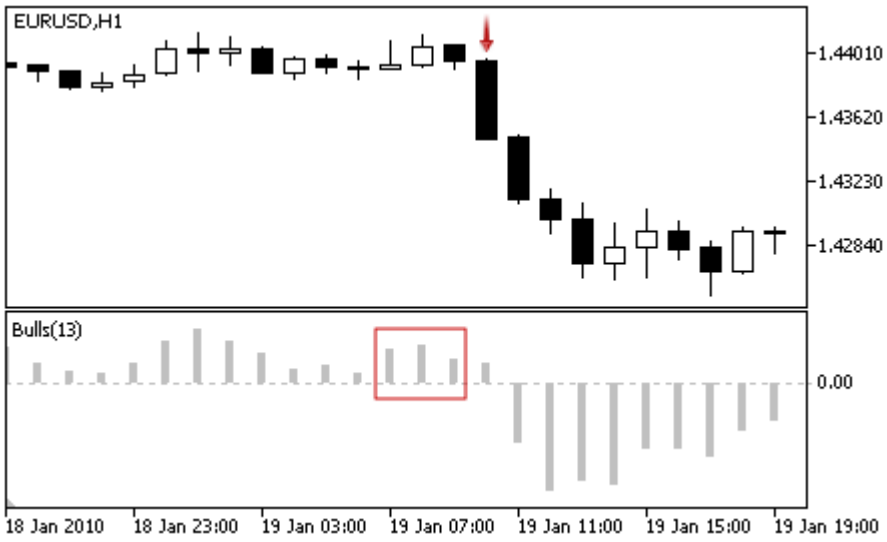
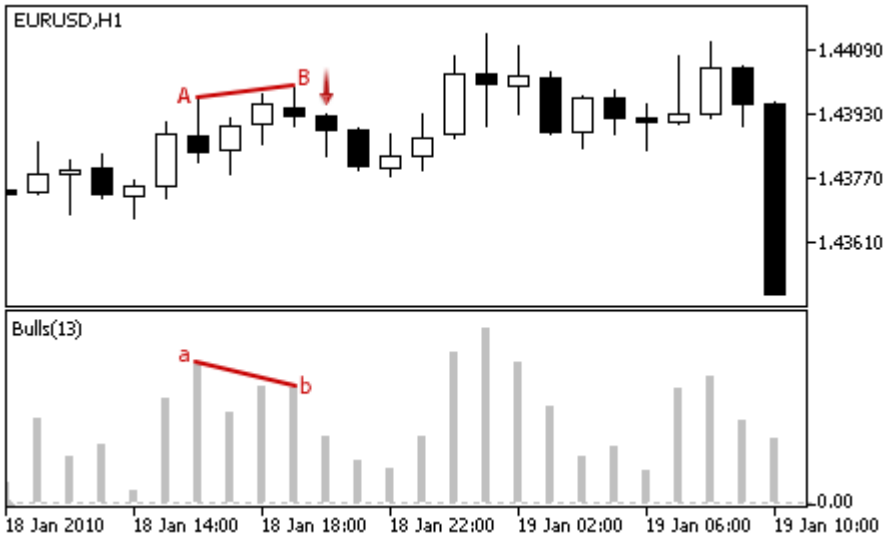
© 2000-2025, MetaQuotes Ltd.

# 多头力度振荡器信号

该信号模块基于 多头力度 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | 无做多信号。 |
| 对于做空 | • 反转 —振荡器返身向下，而它在所分析的柱线上的值高于 0。<br>  <br>• 背离 —振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。此外，振荡器必须不能跌过零轴。<br>  |
| 不反对做多 | 无信号。 |
| 不反对做空 | 振荡器的值大于 0。 |

© 2000-2025, MetaQuotes Ltd.

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| PeriodBulls | 振荡器计算的周期。 |

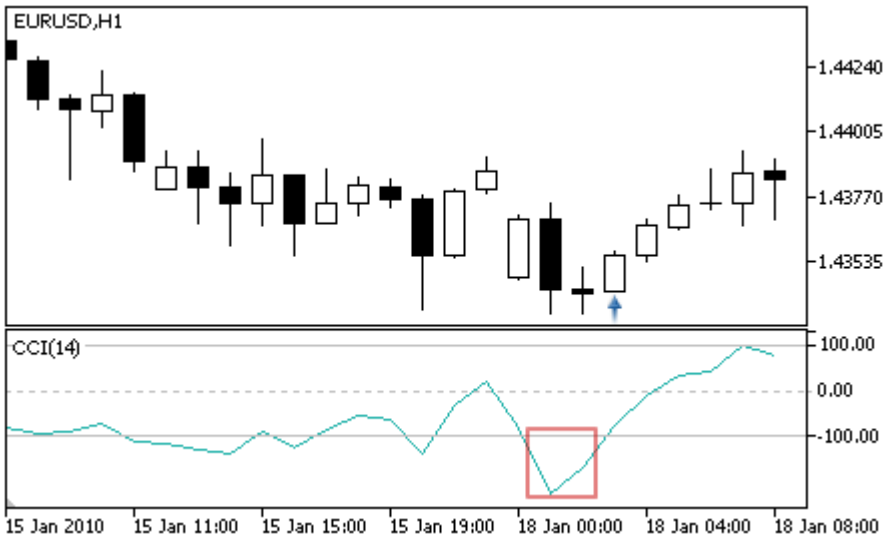
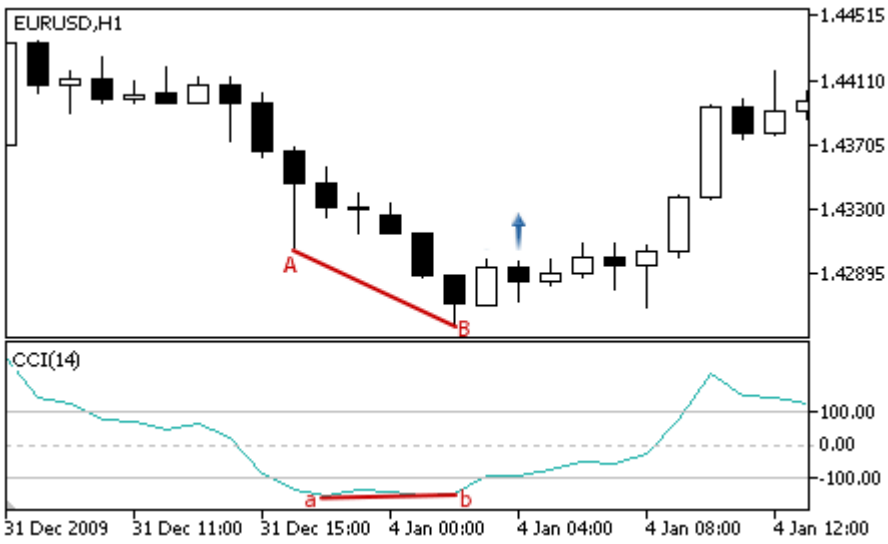
© 2000-2025, MetaQuotes Ltd.

# 商品通道指数振荡器信号

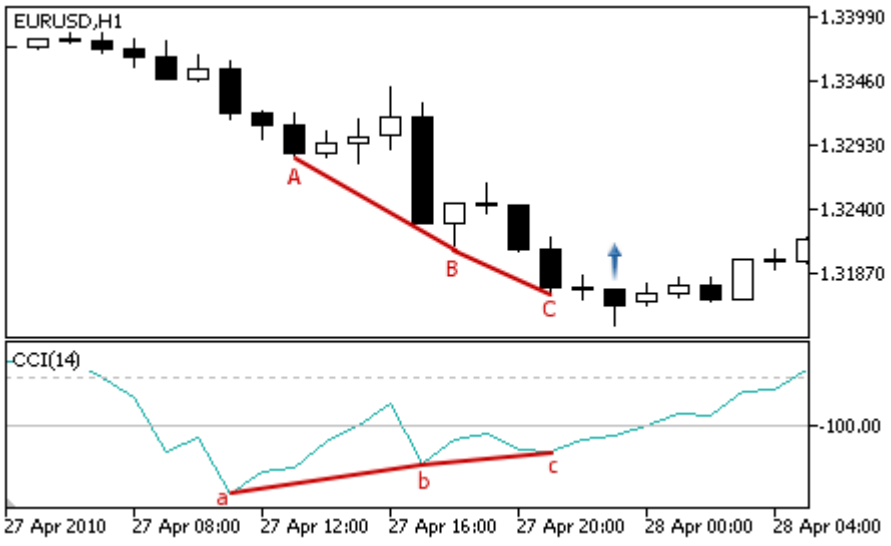
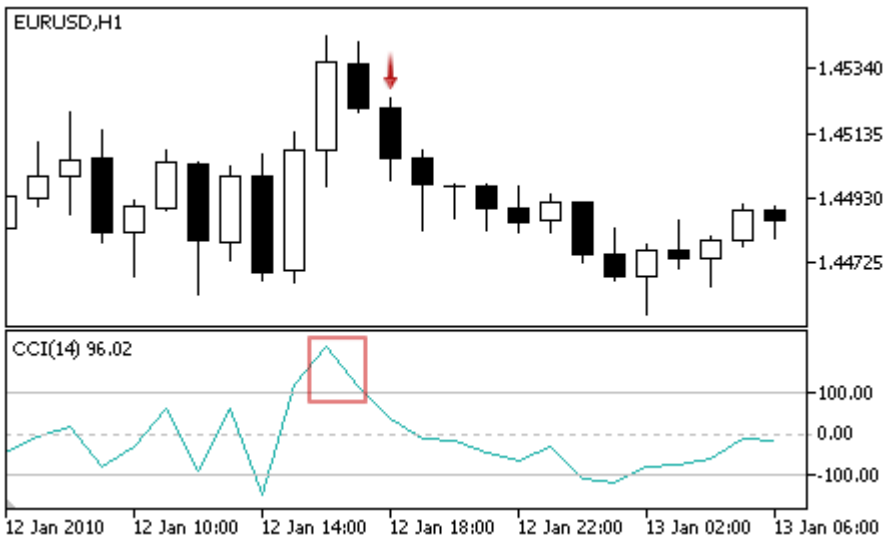
该信号模块基于 商品通道指 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

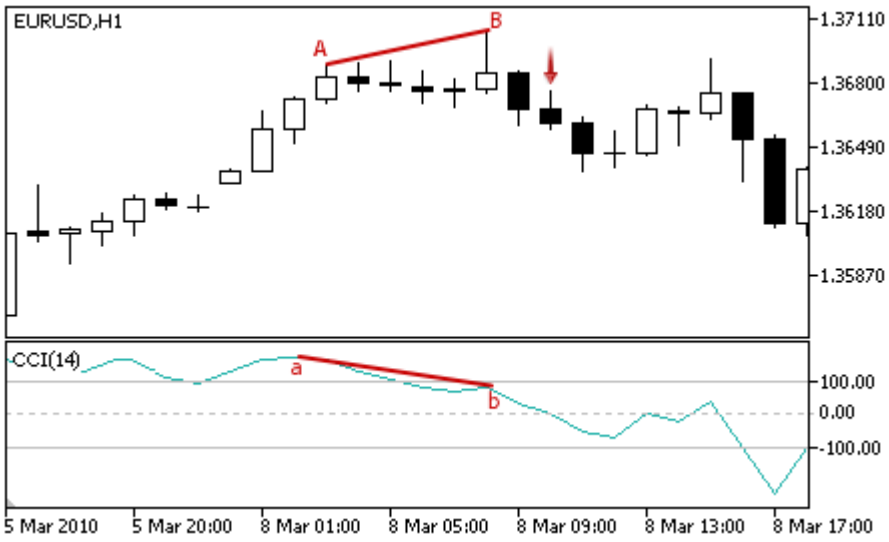
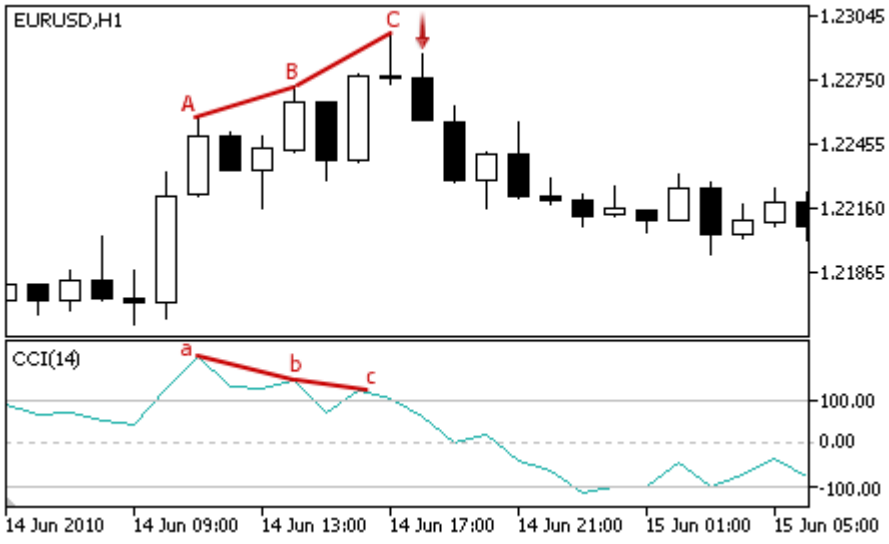
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **超卖之后的反转** — 振荡器返身向上，而它在所分析的柱线上的值高于超卖级别(省缺值是 -100)。<br>  <br>• **背离** — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。<br>  <br>• **双背离** — 振荡器形成三个连续底部，每个底部都高出前一个；并且价格形成三个相应的底部，每个底部都低于前一个。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • **超买之后的反转** — 振荡器返身向下，而它在所分析的柱线上的值低于超卖级别(省缺值是 100)。<br>  <br>• **背离** — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续峰值，每个峰值都低于前一个；并且价格形成三个相应的峰值，每个峰值都高出前一个。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodCCI | 振荡器计算的周期。 |
| Applied | 用于振荡器计算的 价格序列 。 |

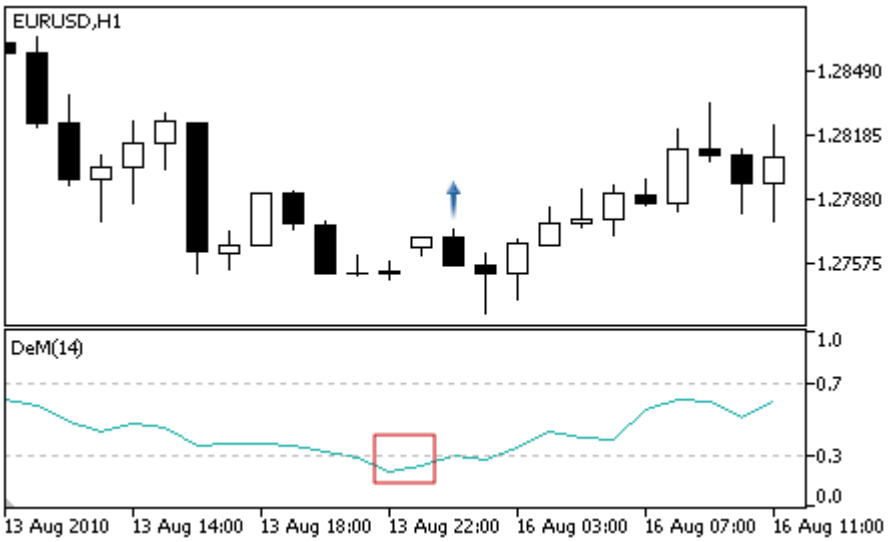
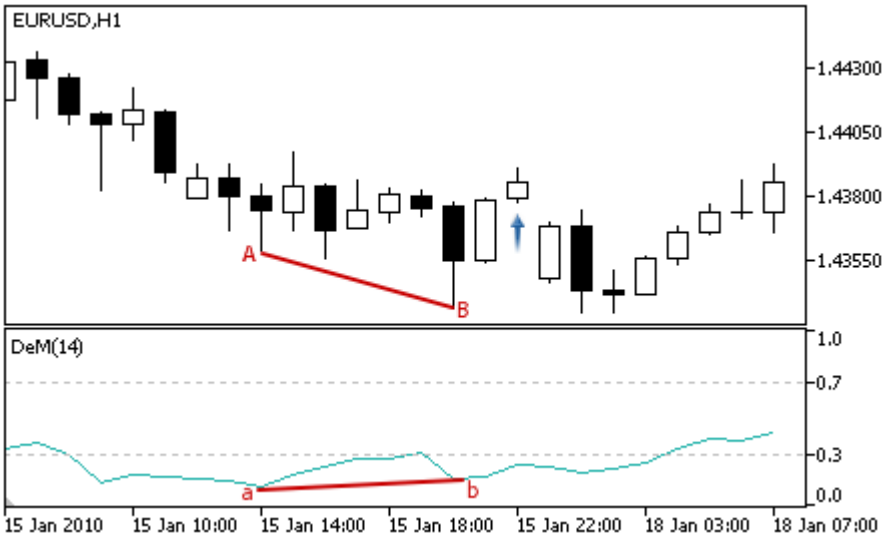
© 2000-2025, MetaQuotes Ltd.

# DeMarker 振荡器信号

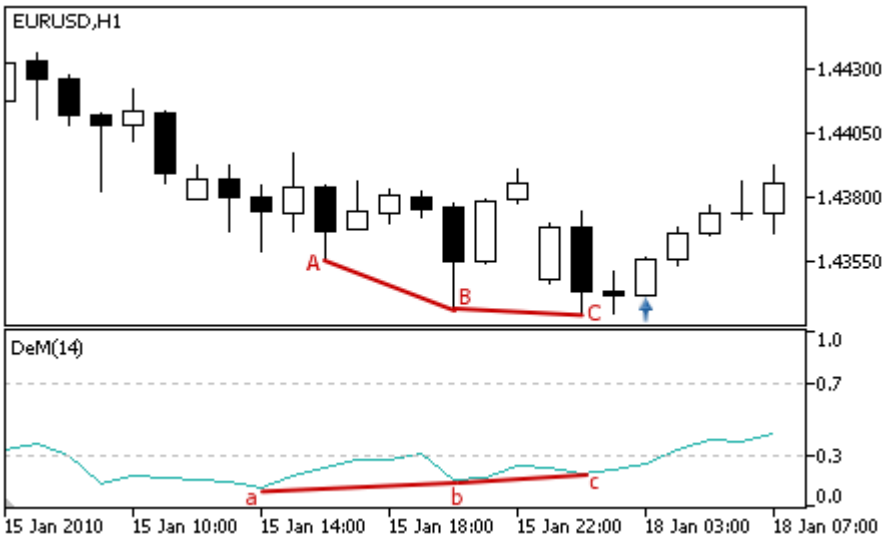
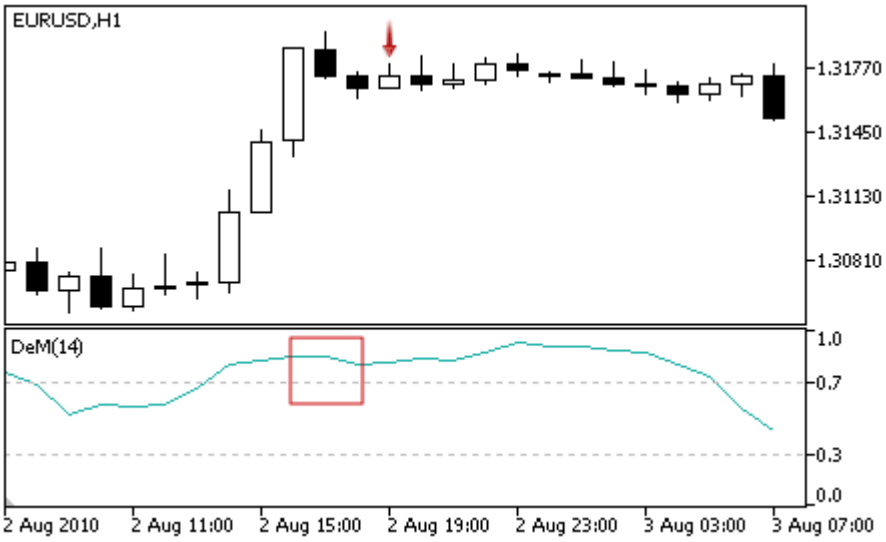
该信号模块基于 DeMarker 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

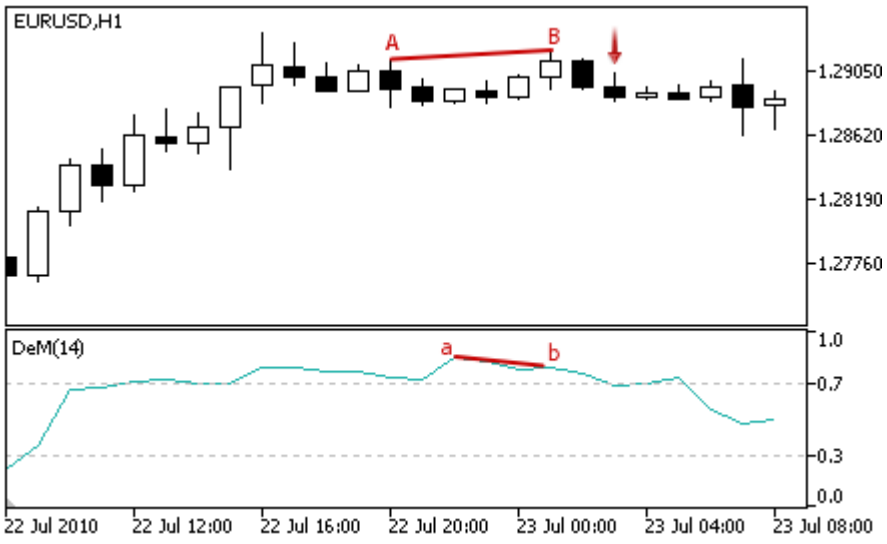
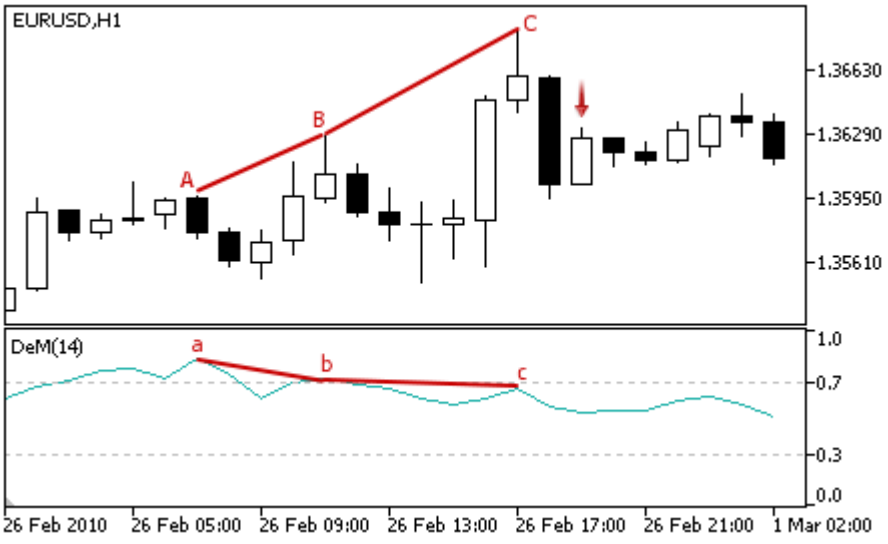
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **超卖之后的反转** —振荡器返身向上，而它在所分析的柱线上的值高于超卖级别(省缺值是 0.3)。<br>  <br><br>• **背离** — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。<br>  <br><br>• **双背离** —振荡器形成三个连续底部，每个底部都高出前一个；并且价格形成三个相应的底部，每个底部都低于前一个。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 超买之后的反转 — 振荡器返身向下，而它在所分析的柱线上的值低于超买级别(省缺值是 0.7)。<br>  <br>• 背离 — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续峰值，每个峰值都低于前一个；并且价格形成三个相应的峰值，每个峰值都高出前一个。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodDeM | 振荡器计算的周期。 |

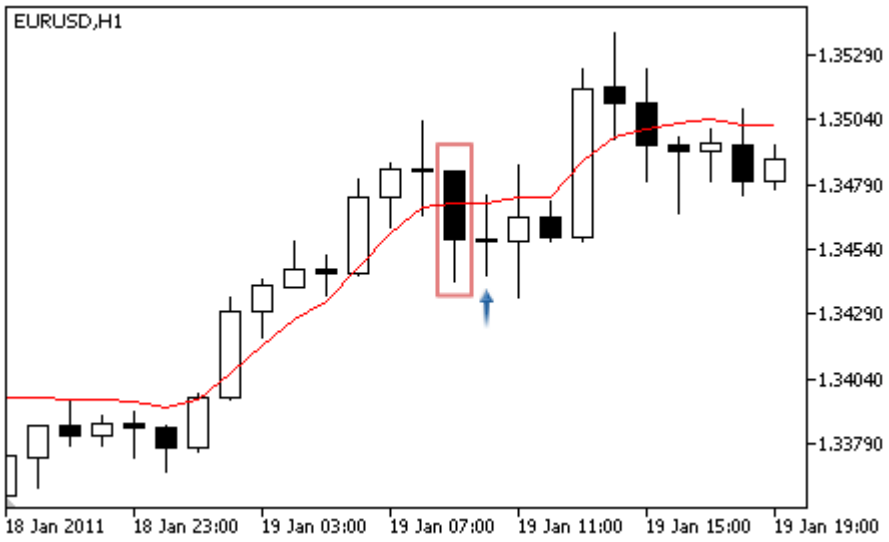
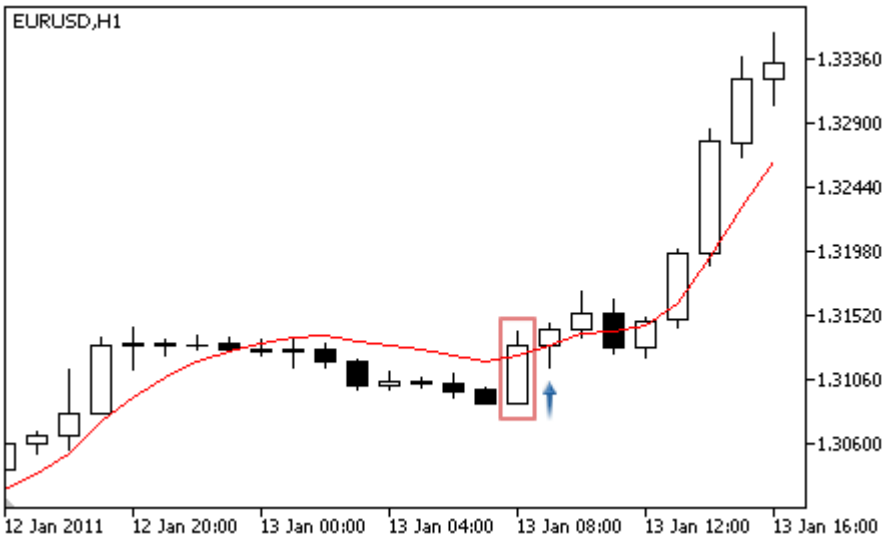
© 2000-2025, MetaQuotes Ltd.

# 双重指数移动均线指标信号

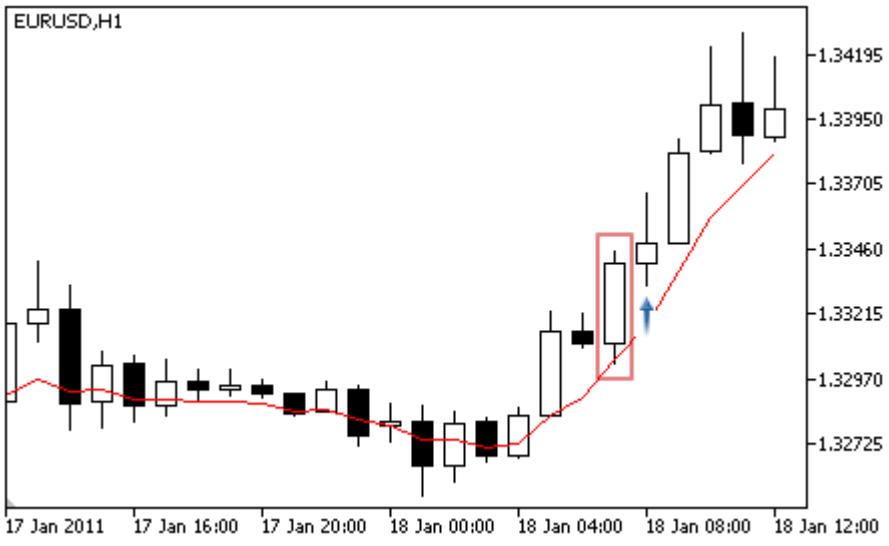
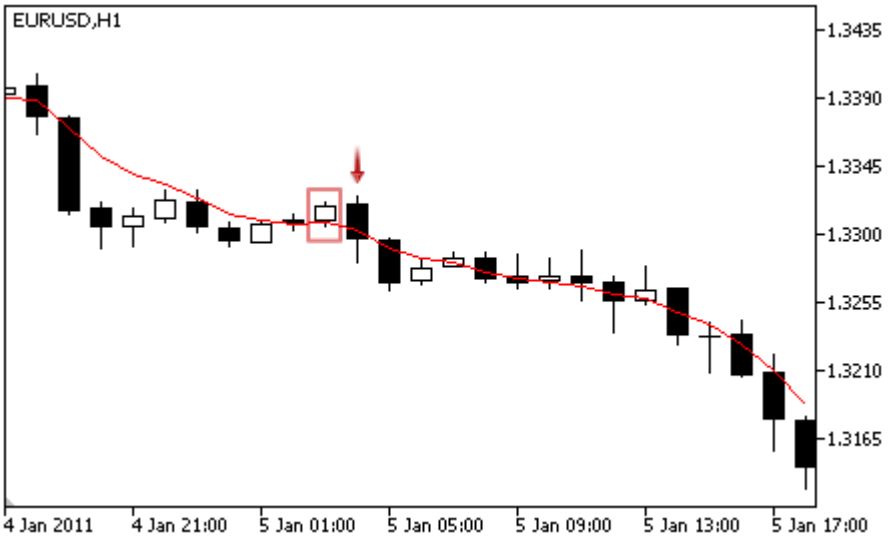
该信号模块基于 双重指数移动均线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

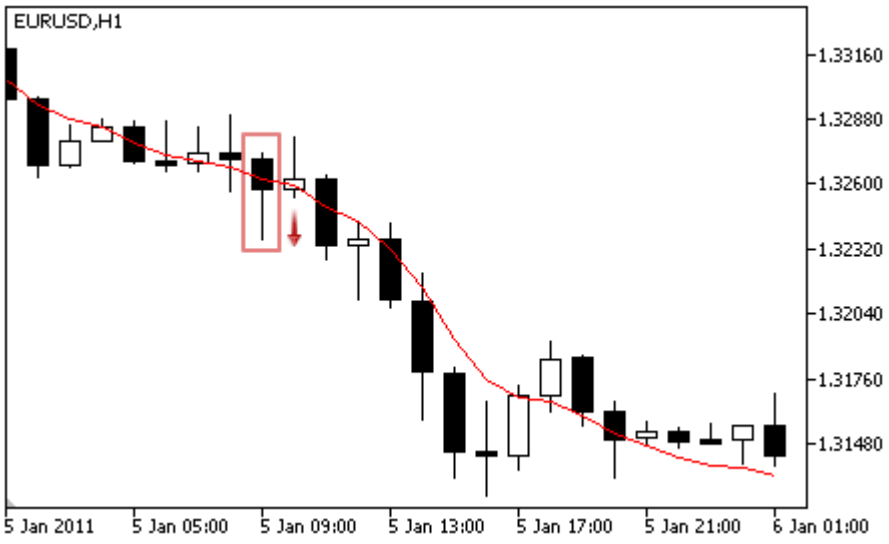
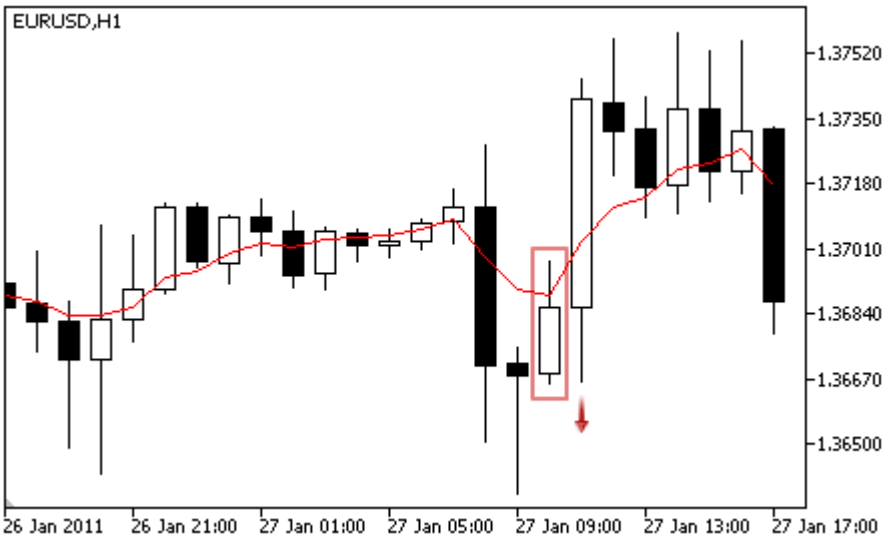
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标上涨（弱信号）。<br>  <br><br>• 移动均线穿越。价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标上涨（强信号）。<br>  <br><br>• 柱线的下影线穿越指标（所分析柱线的开盘价和收盘价均高于指标，而最低价低于指标）且指标上涨（弱信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标下跌（弱信号）。<br>  <br>• 移动均线穿越。价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标下跌（强信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 柱线的上影线穿越指标（所分析柱线的开盘价和收盘价均低于指标，而最高价高于指标）且指标下跌（弱信号）。<br>  |
| 不反对做多 | 价格高于指标。 |
| 不反对做空 | 价格在指标之下。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 均线指标的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |

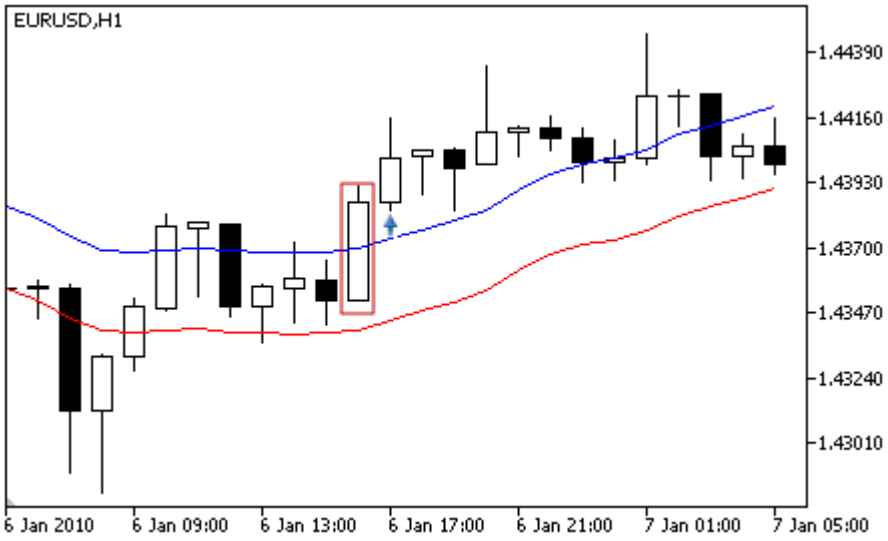
© 2000-2025, MetaQuotes Ltd.

# 包络线指标信号

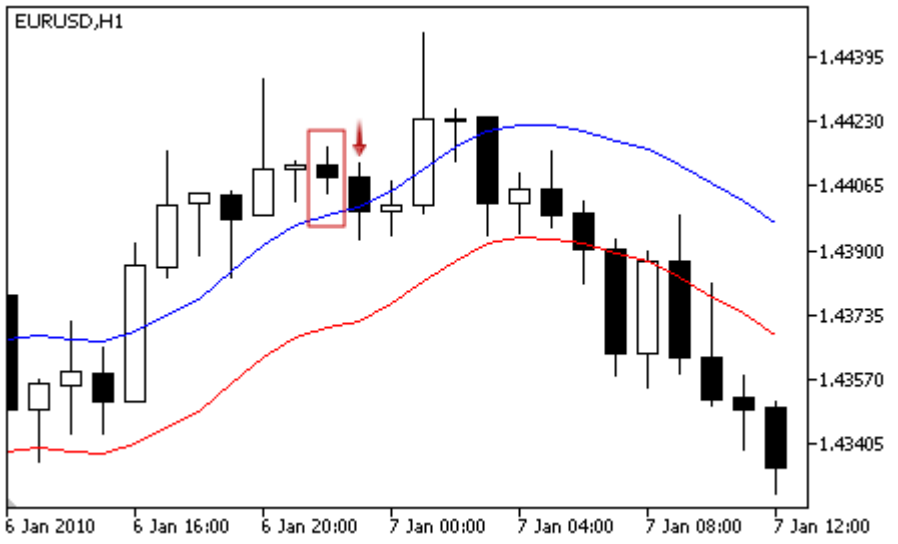
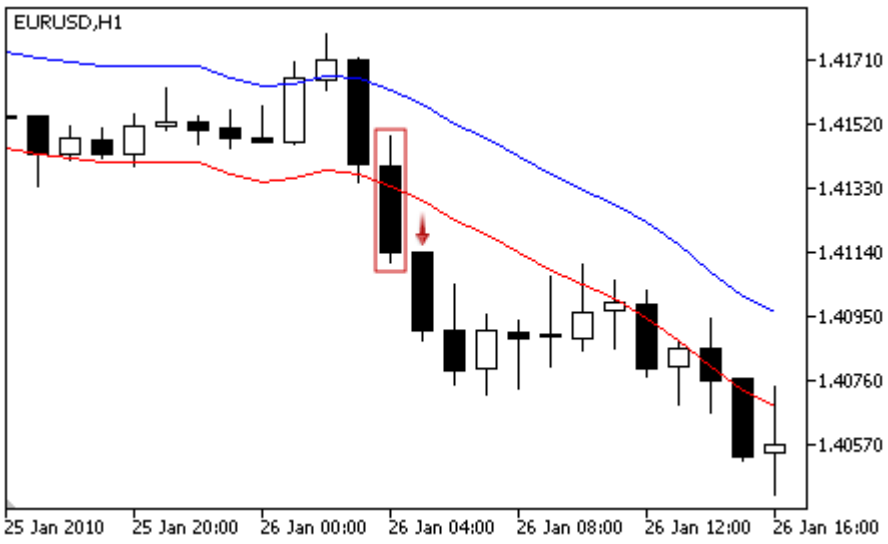
该信号模块基于 包络线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格在所分析的柱线中接近指标的低位线。<br> <br><br>• 价格在所分析的柱线中穿越指标的上位线。<br>  <br> |
| 对于做空 | • 价格在所分析的柱线中接近指标的上位线。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 价格在所分析的柱线中穿越指标的低位线。<br>  |
| 不反对做多 | 无信号。 |
| 不反对做空 | 无信号。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 指标计算的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |
| Deviation | 包络线边界距中线（MA）的偏差百分比。 |

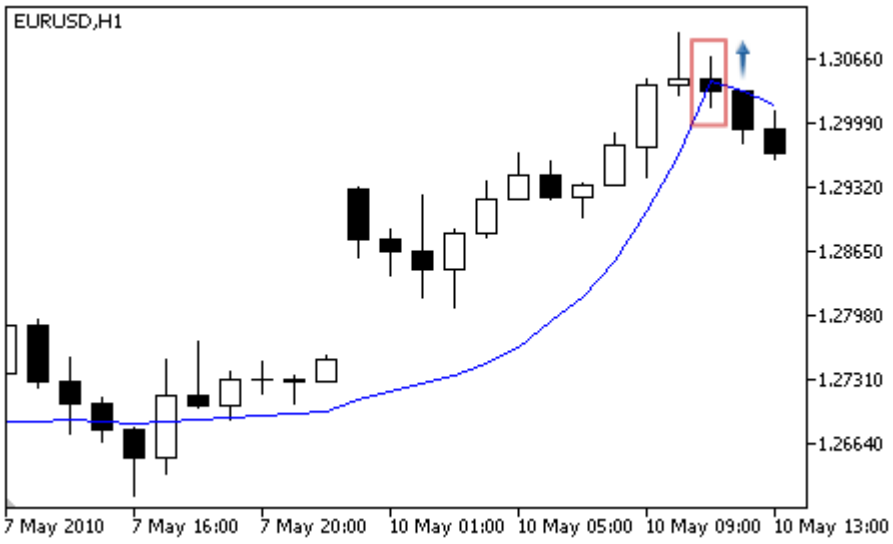
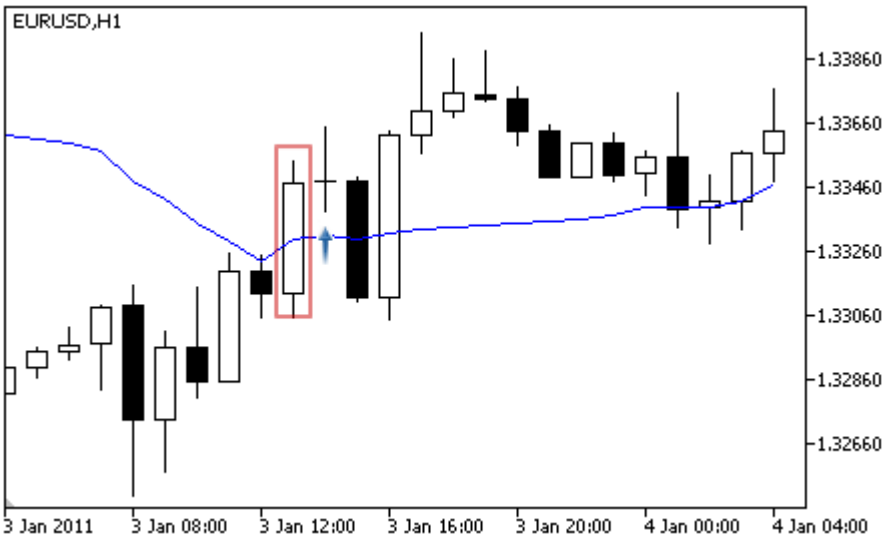
© 2000-2025, MetaQuotes Ltd.

# 分形自适应移动均线指标信号

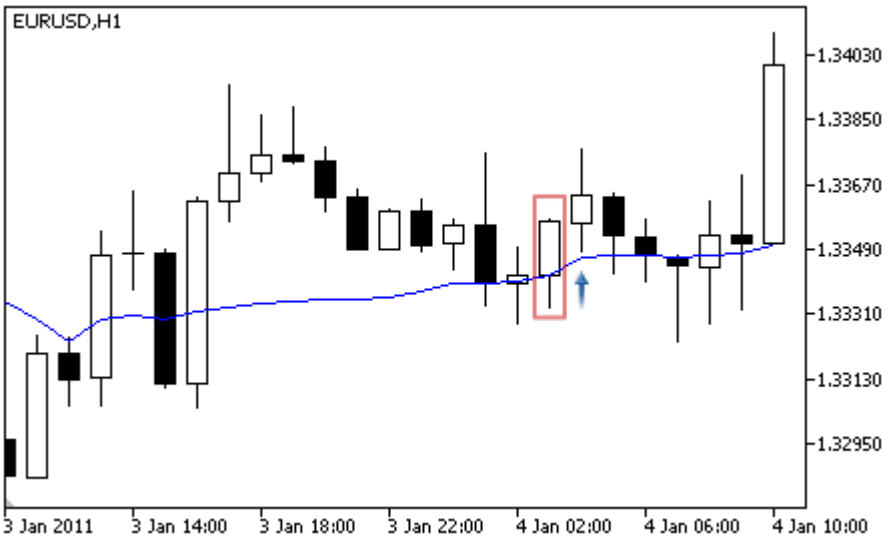
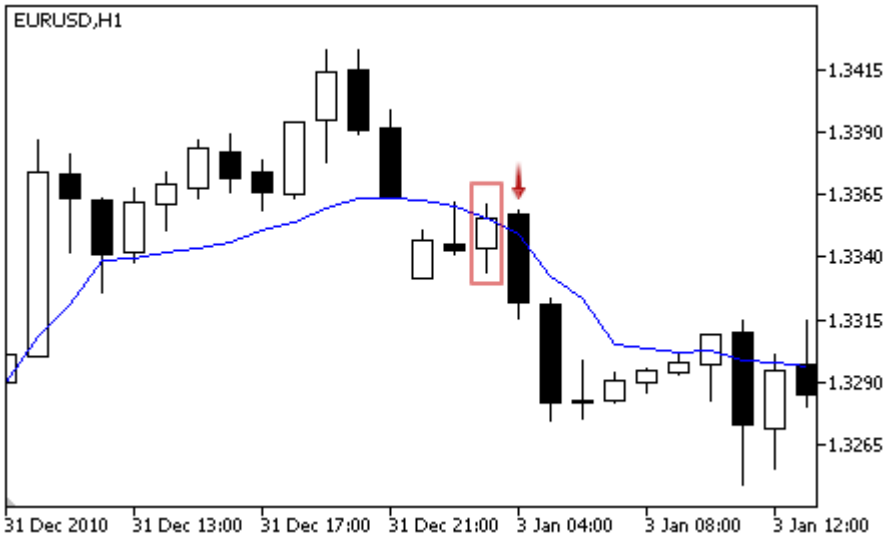
该信号模块基于 分形自适应移动均线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

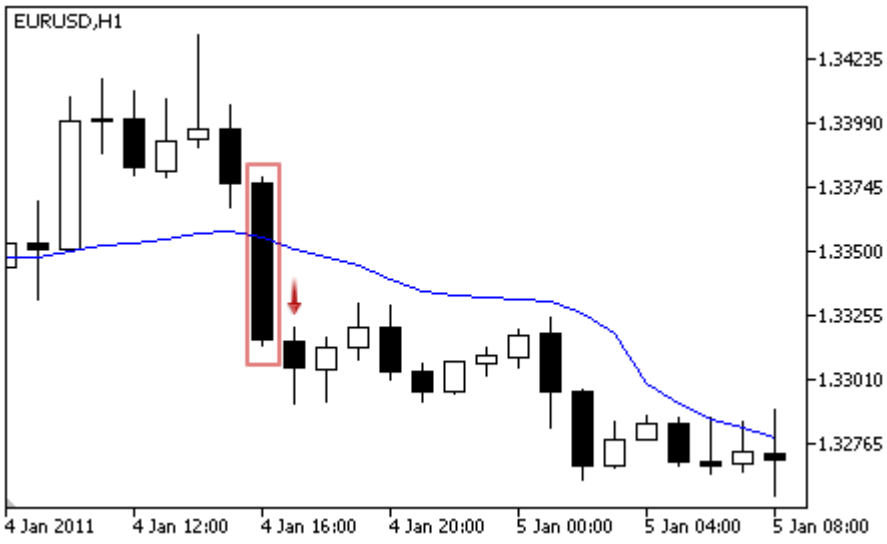
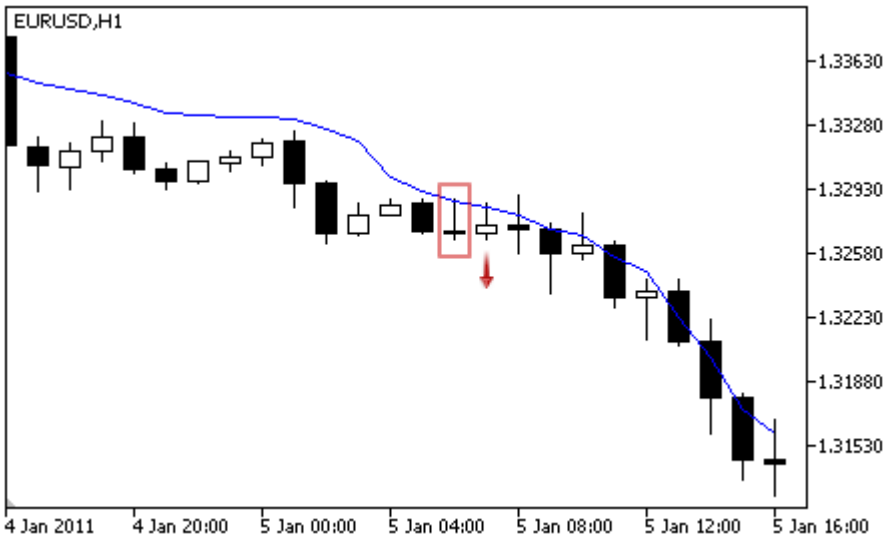
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标上涨（弱信号）。<br>  <br><br>• **移动均线穿越。**价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标上涨（强信号）。<br>  <br><br>• 柱线的下影线穿越指标（所分析柱线的开盘价和收盘价均高于指标，而最低价低于指标）且指标上涨（弱信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标下跌（弱信号）。<br>  <br>• **移动均线穿越**。价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标下跌（强信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 柱线的上影线穿越指标（所分析柱线的开盘价和收盘价均低于指标，而最高价高于指标）且指标下跌（弱信号）。<br>  |
| 不反对做多 | 价格在指标之上。 |
| 不反对做空 | 价格在指标之下。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 均线指标的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |

© 2000-2025, MetaQuotes Ltd.

# 日内时间过滤信号

该模块基于市场模型在单位时间内的变化率的假设。使用此模块，您能够用小时和周内日期来过滤从其它模块接收的信号。它可以提高生成信号的品质，因为消减了不受欢迎的时间周期。在模块信号基础上制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | 无信号。 |
| 对于做空 | 无信号。 |
| 不反对做多 | 当前日期和时间符合指定参数。 |
| 不反对做空 | 当前日期和时间符合指定参数。 |

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| GoodHourOfDay | 仅在指定日内小时内（从 0 至 23）时，交易信号将被启用。如果值为 -1，信号将会全天启用。 |
| BadHoursOfDay | 位字段。每个位字段对应一个日内小时（0 位 - 0 hour，...，23 位 - 第 23 小时）。如果位值等于 0，交易信号在相应的整个小时启用。如果位值等于 1，交易信号在相应的整个小时禁用。指定数字呈现为二进制数，且用作位掩码。<br>禁用小时的优先权高于启用小时。 |
| GoodDayOfWeek | 仅在指定周内日期（从 0 至 6）时，交易信号将被启用。如果值为 -1，信号将会所有天启用。 |
| BadDaysOfWeek | 位字段。每个位字段对应一个周内日期（0 位 - 周日，...，6 位 - 周六）。如果位值等于 0，交易信号在相应的整日启用。如果位值等于 1，交易信号在相应的整日禁用。指定数字呈现为二进制数，且用作位掩码。<br>禁用日的优先权高于启用日。 |

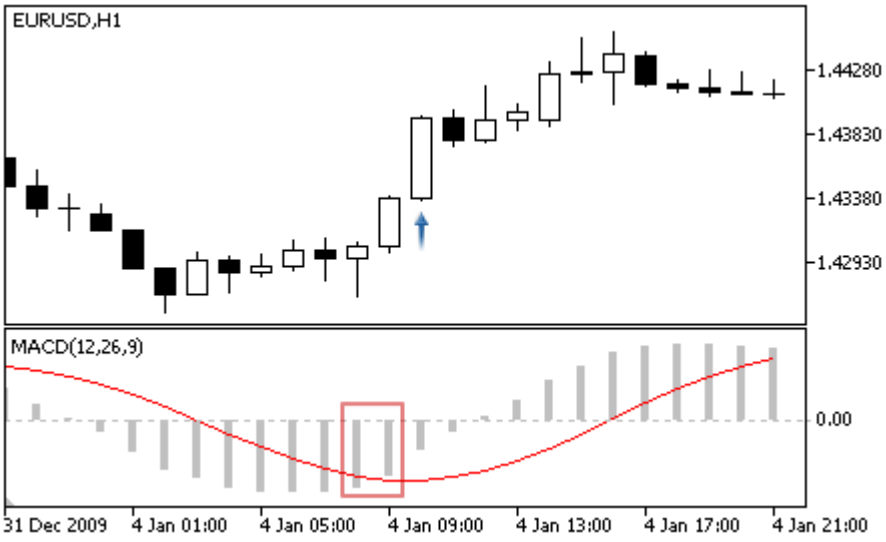
© 2000-2025, MetaQuotes Ltd.

# MACD 振荡器信号

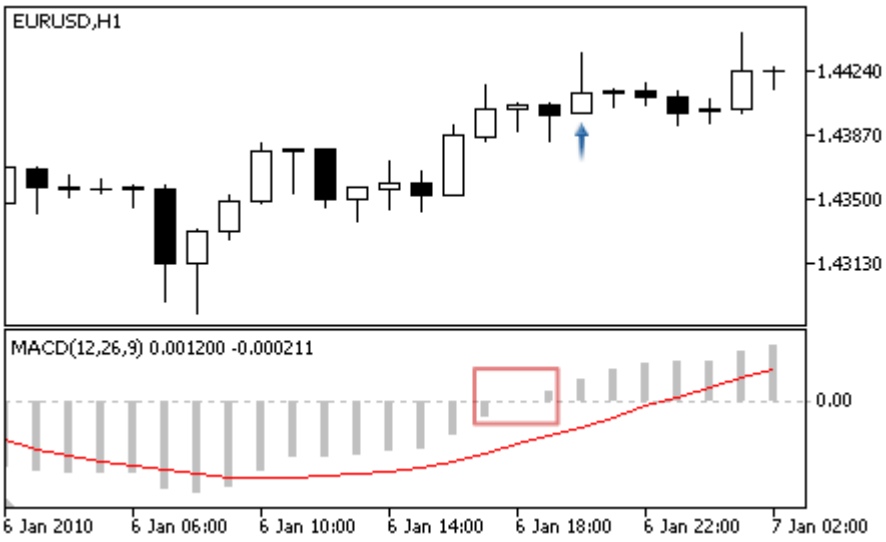
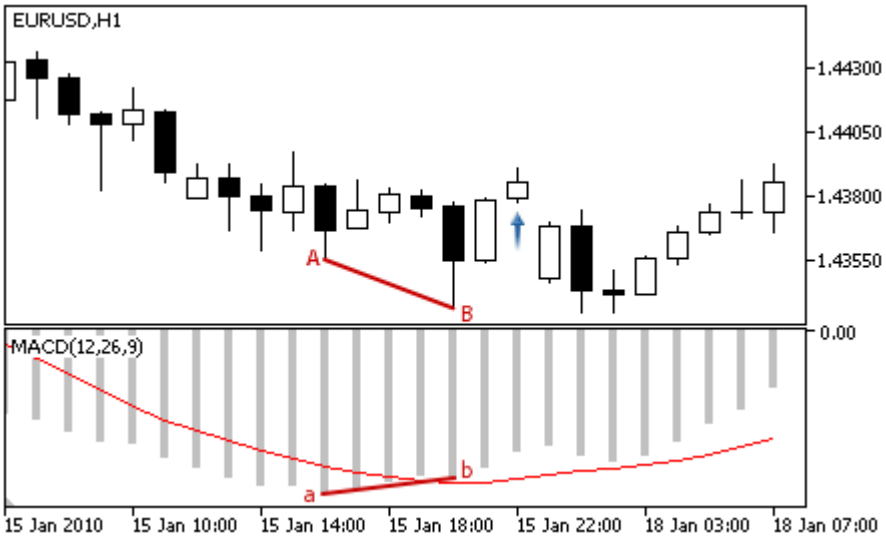
该信号模块基于 MACD 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

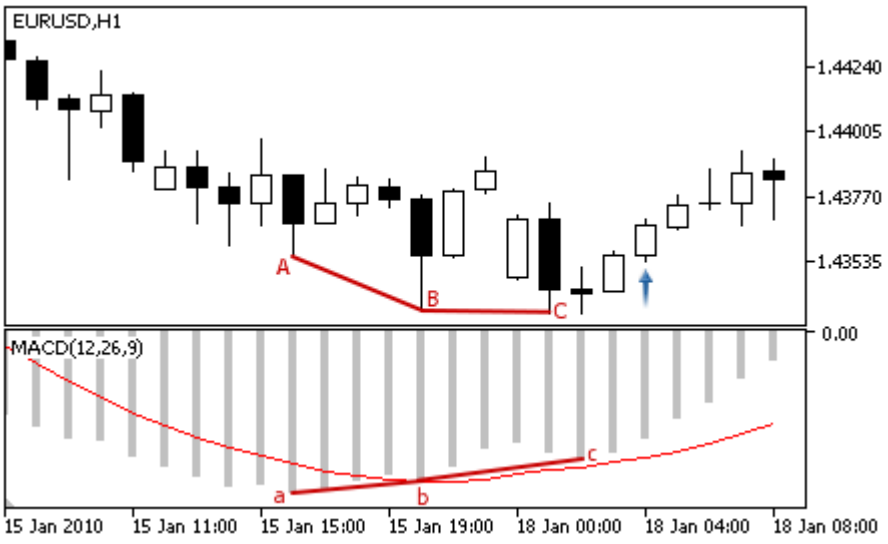
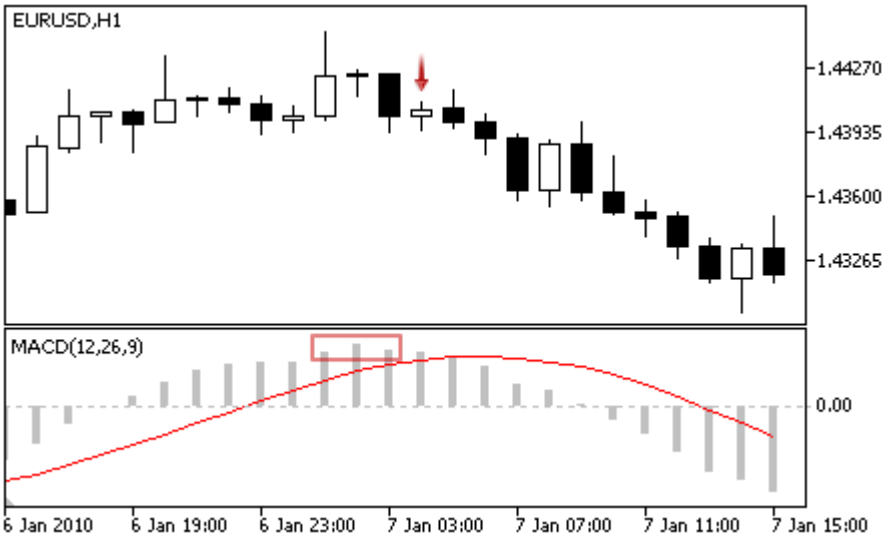
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **反转** —振荡器返身向上（振荡器在所分析的柱线里上升，而在前一根线里下降）。 <br>• **主线穿越信号线** —在所分析的柱线里，主线在信号线之上，且在前一根线里低于信号线。  <br>• **穿越零轴** —主线在所分析的柱线上高于零轴，且在前根柱线上低于零轴。 |

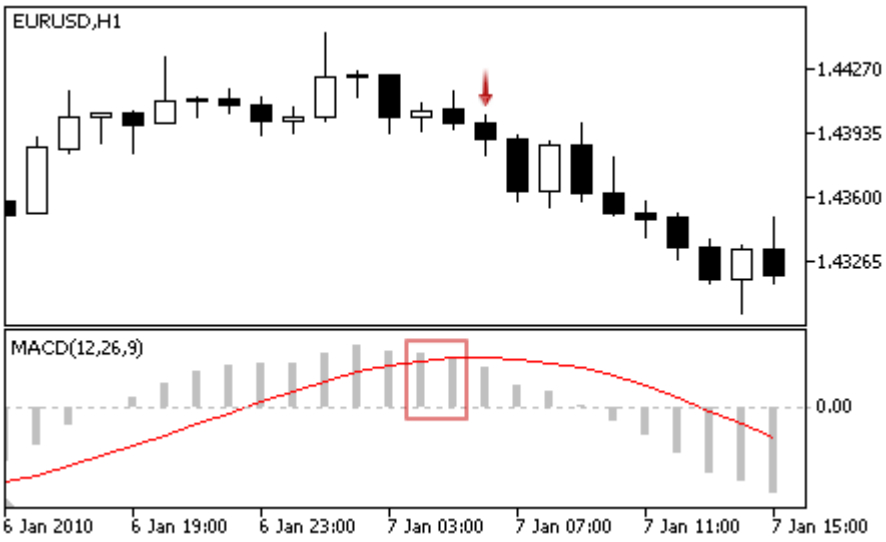
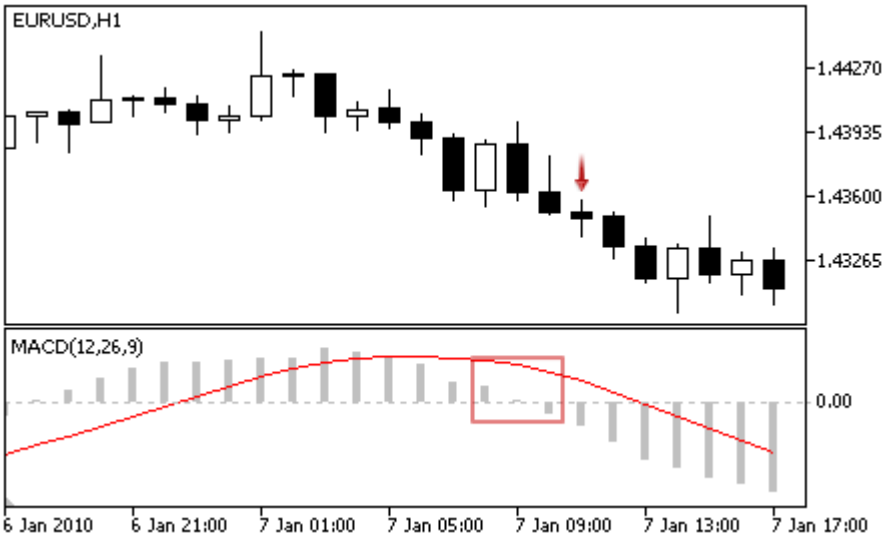
© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 背离 — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。<br>  <br>• 双背离 —振荡器形成三个连续底部，每个底部都高出前一个；并且价格形成三个相应的底部，每个底部都低于前一个。 |

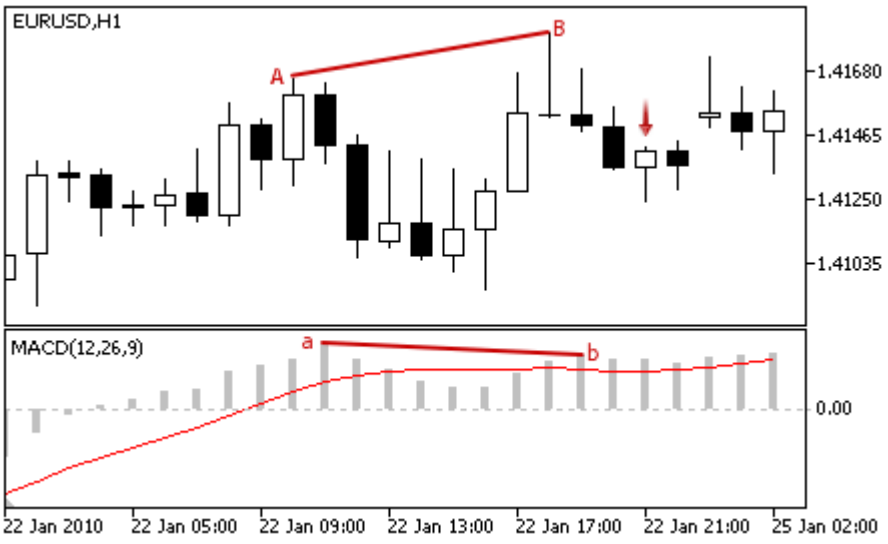
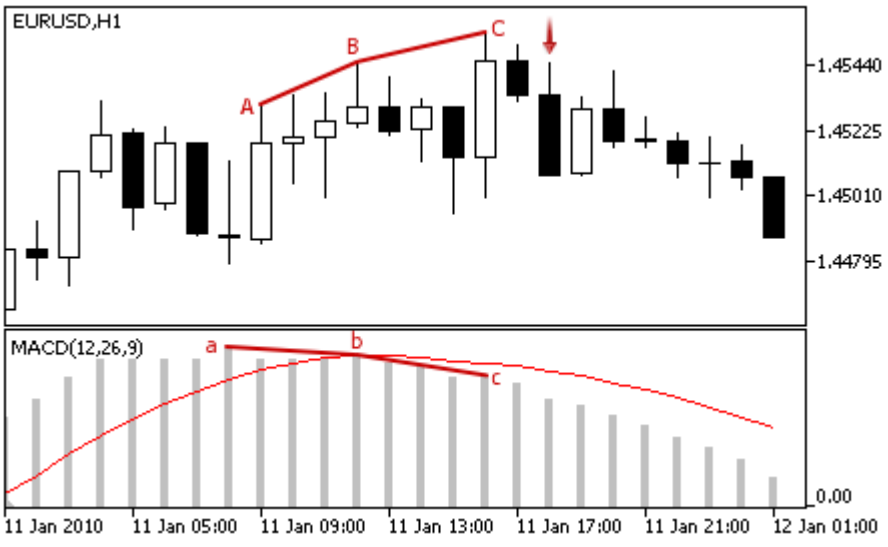
© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 反转 —振荡器返身向下（振荡器在所分析的柱线里下降，而在前一根线里上升）。<br>  <br>• 主线穿越信号线 —在所分析的柱线里，主线在信号线之下，且在前一根线里高于信号线。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 穿越零轴 — 主线在所分析的柱线上低于零轴，且在前根柱线上高于零轴。<br>  <br>• 背离 — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续峰值，每个峰值都低于前一个；并且价格形成三个相应的峰值，每个峰值都高出前一个。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodFast | 快速 EMA 计算周期。 |
| PeriodSlow | 慢速 EMA 计算周期。 |
| PeriodSignal | 平滑周期。 |
| Applied | 用于振荡器计算的 价格序列 。 |

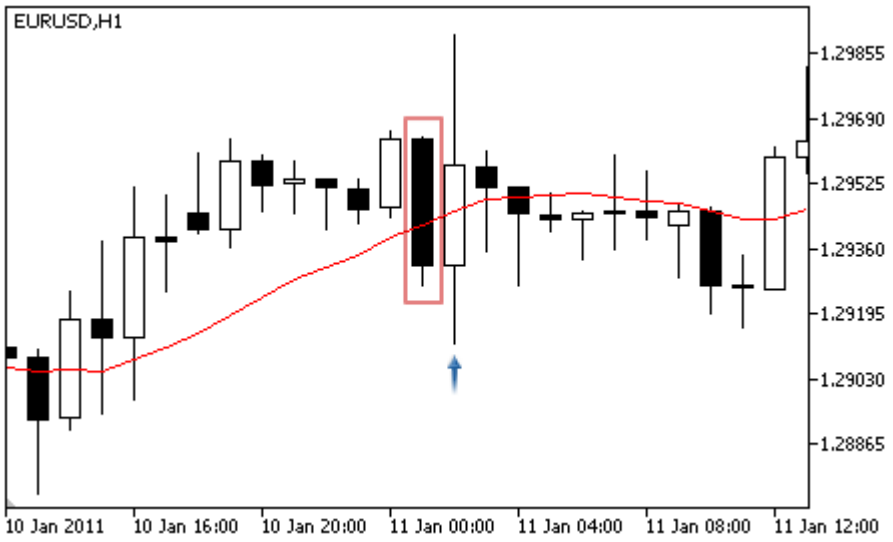
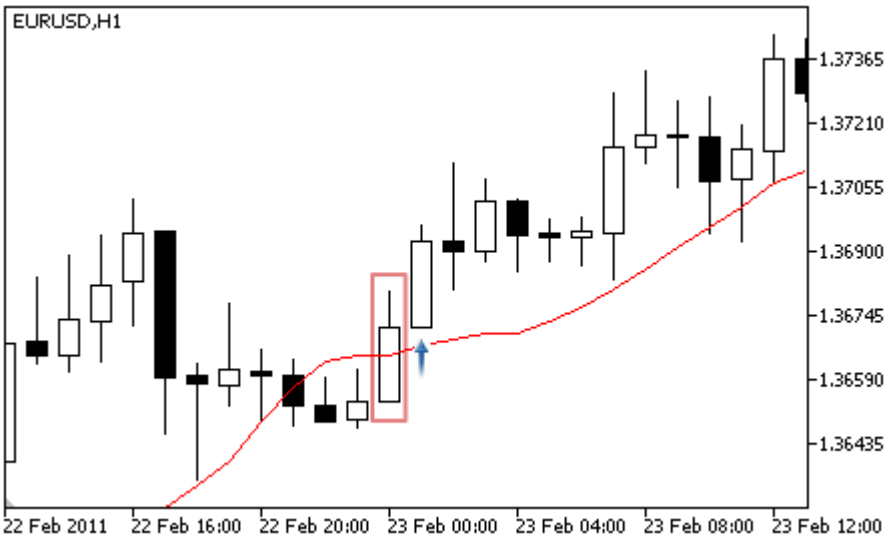
© 2000-2025, MetaQuotes Ltd.

# 移动均线指标信号

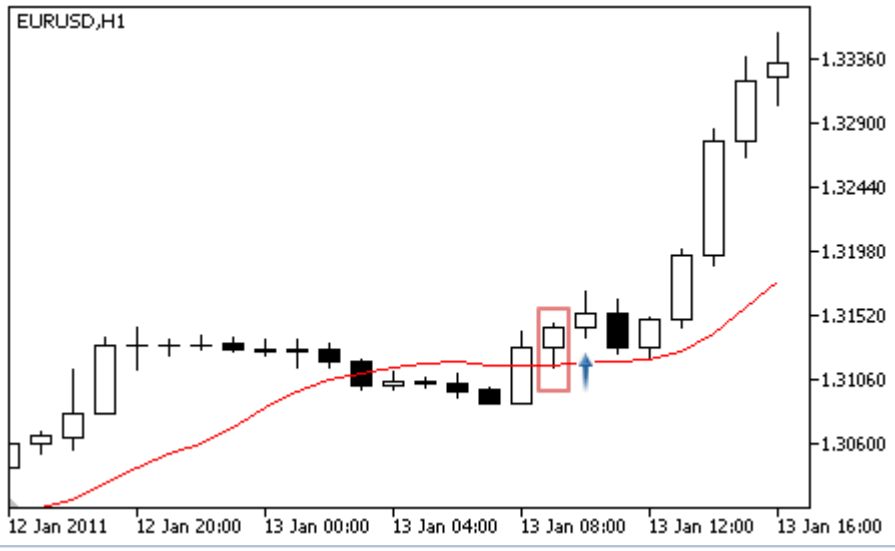
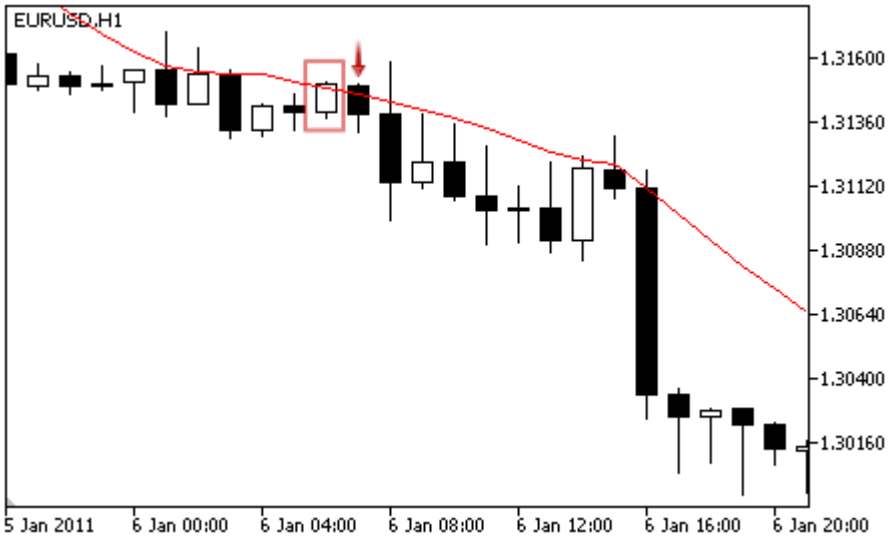
该信号模块基于 移动均线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

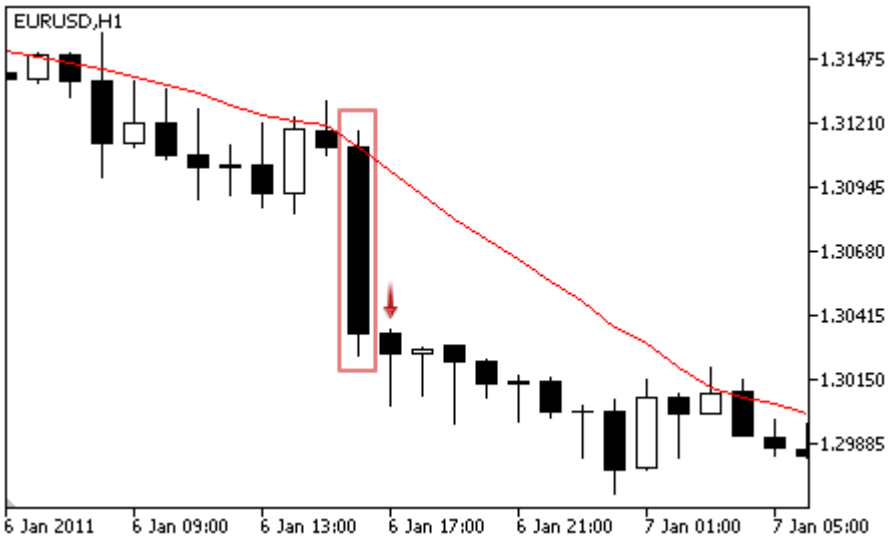
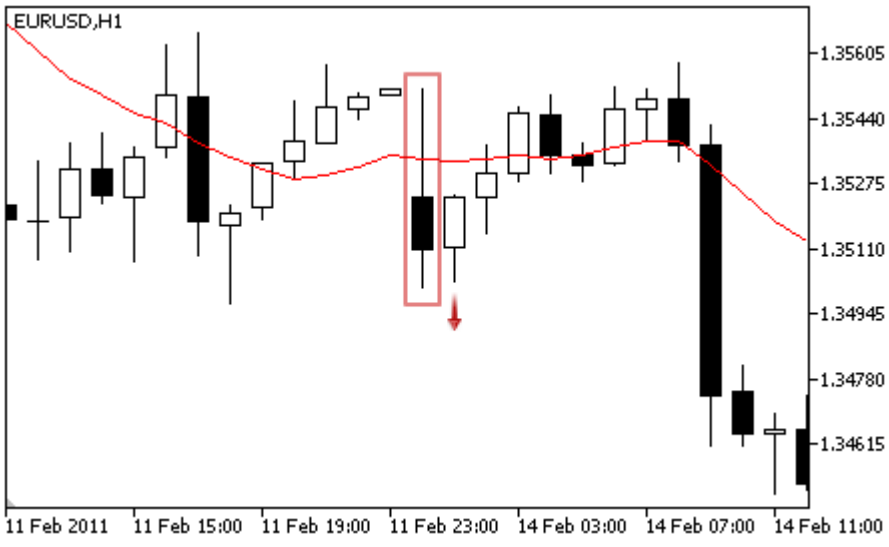
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标上涨（弱信号）。<br>  <br><br>• 移动均线穿越。价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标上涨（强信号）。<br>  <br><br>• 柱线的下影线穿越指标（所分析柱线的开盘价和收盘价均高于指标，而最低价低于指标）且指标上涨（弱信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标下跌（弱信号）。<br>  <br>• **移动均线穿越**。价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标下跌（强信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 柱线的上影线穿越指标（所分析柱线的开盘价和收盘价均低于指标，而最高价高于指标）且指标下跌（弱信号）。<br>  |
| 不反对做多 | 价格在指标之上。 |
| 不反对做空 | 价格在指标之下。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 均线指标的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |

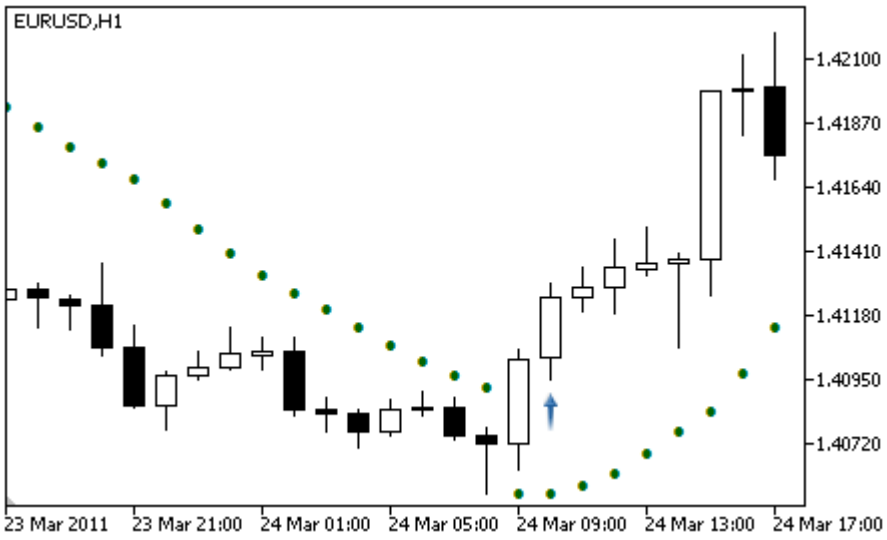
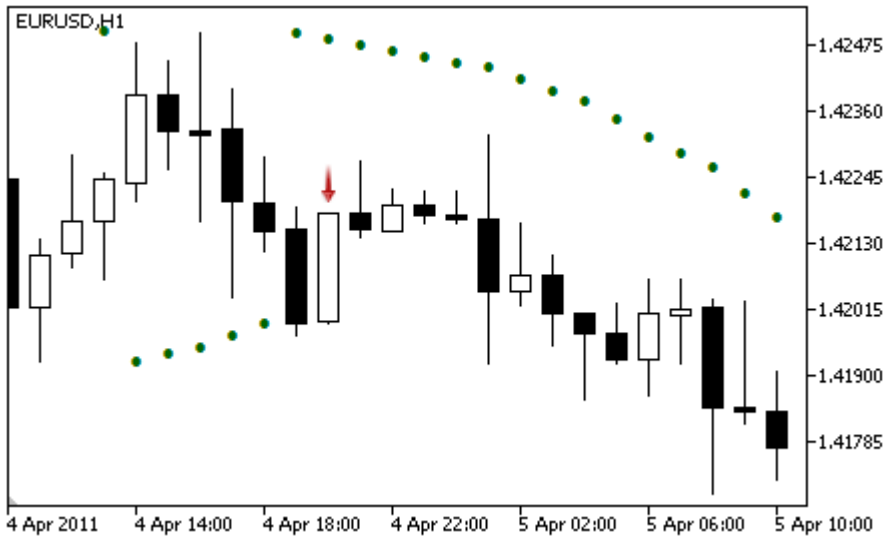
© 2000-2025, MetaQuotes Ltd.

# 抛物线转向指标信号

该信号模块基于 抛物线转向 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | 反转 —指标在所分析的柱线上低于价格，且在前根柱线上高于价格。  |
| 对于做空 | 反转 —指标在所分析的柱线上高于价格，且在前根柱线上低于价格。  |
| 不反对做多 | 价格在指标之上。 |
| 不反对做空 | 价格在指标之下。 |

注释

© 2000-2025, MetaQuotes Ltd.

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| Step | 指标速度的增量。 |
| Maximum | 指标与价格聚合的最大速率。 |

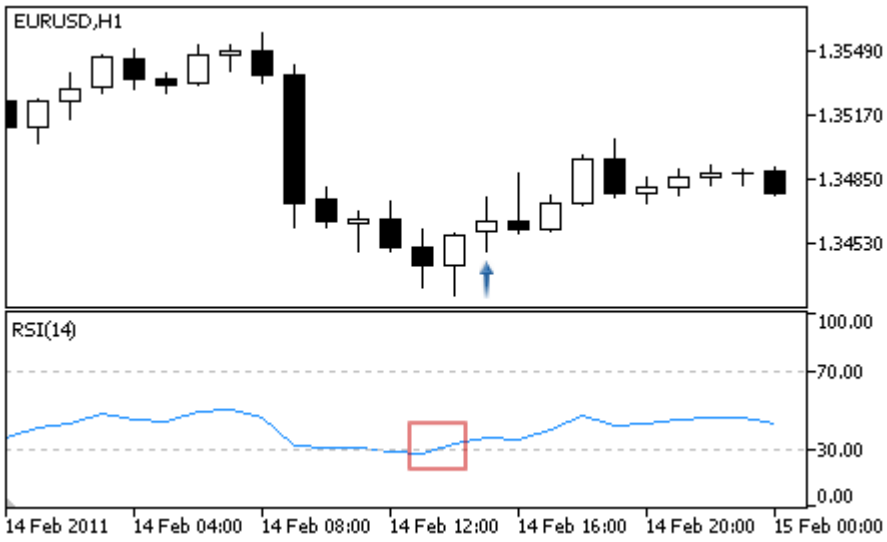
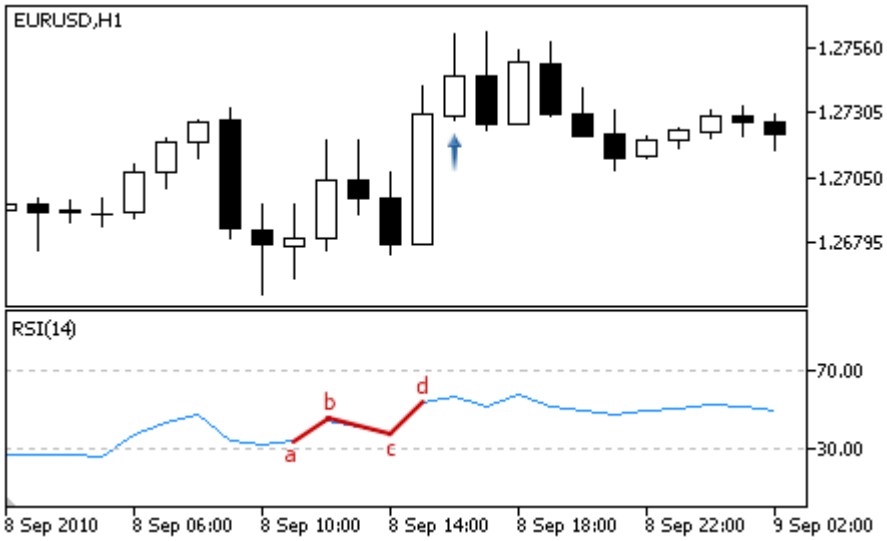
© 2000-2025, MetaQuotes Ltd.

# 相对强度指数振荡器信号

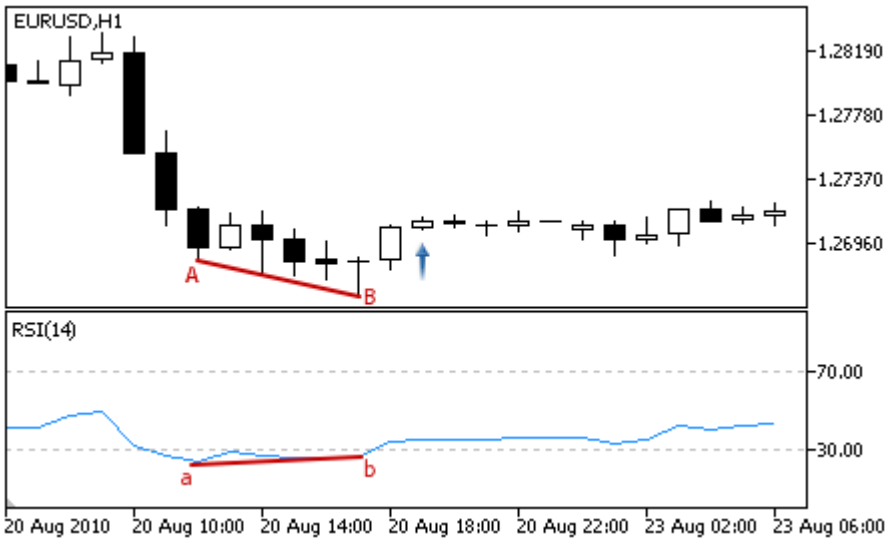
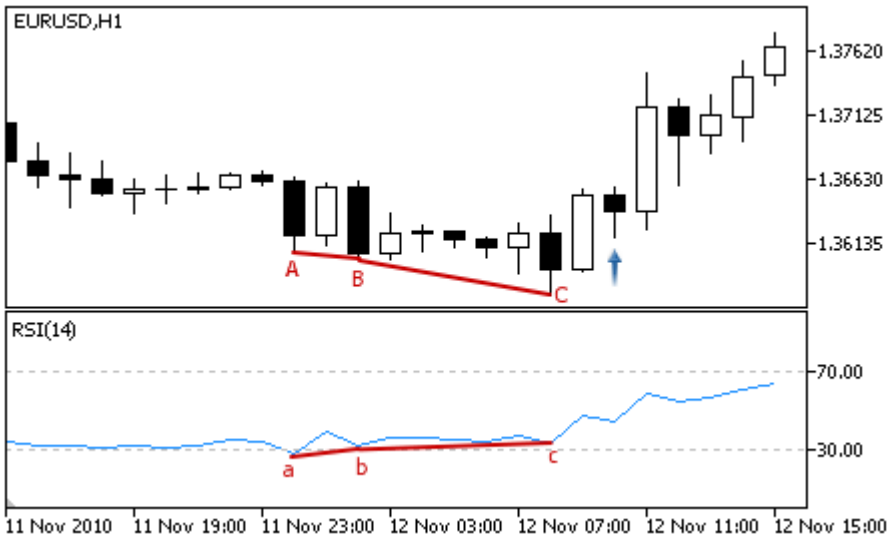
该信号模块基于 Relative Strength Index 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

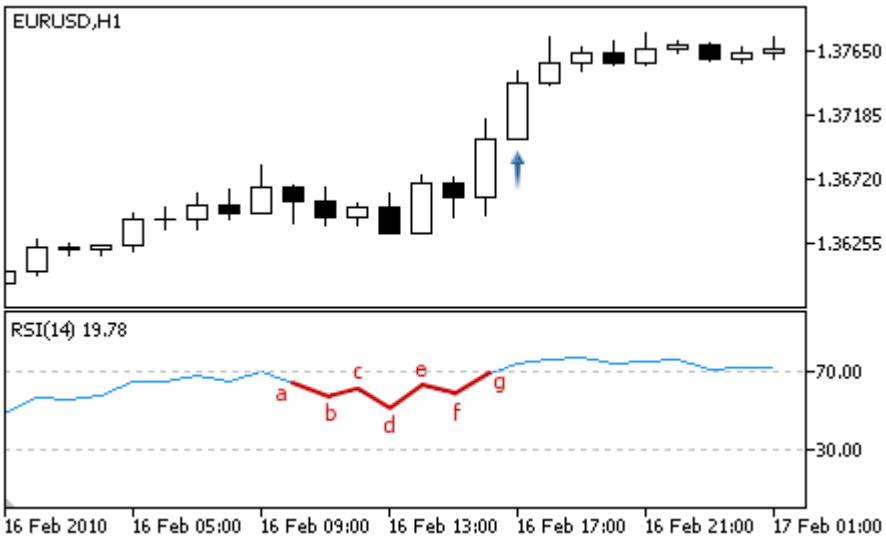
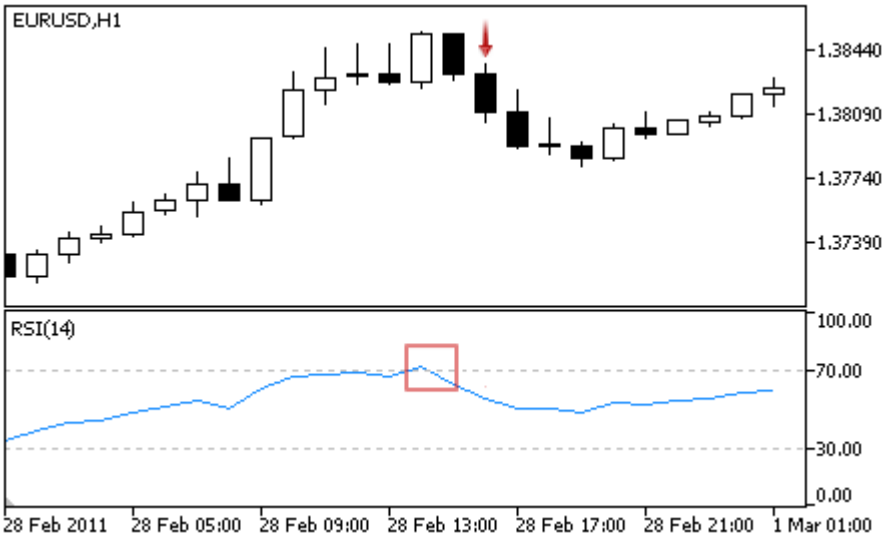
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **超卖之后的反转** — 振荡器返身向上，而它在所分析的柱线上的值高于超卖级别 (省缺值是 30)。<br>  <br>• **波段失败** —振荡器值在所分析的柱线中上涨且高于前一峰值。<br>  <br>• **背离** — 振荡器最近分析的底部比更早的一个要低，而相应的价格波谷比更早的一个要低。 |

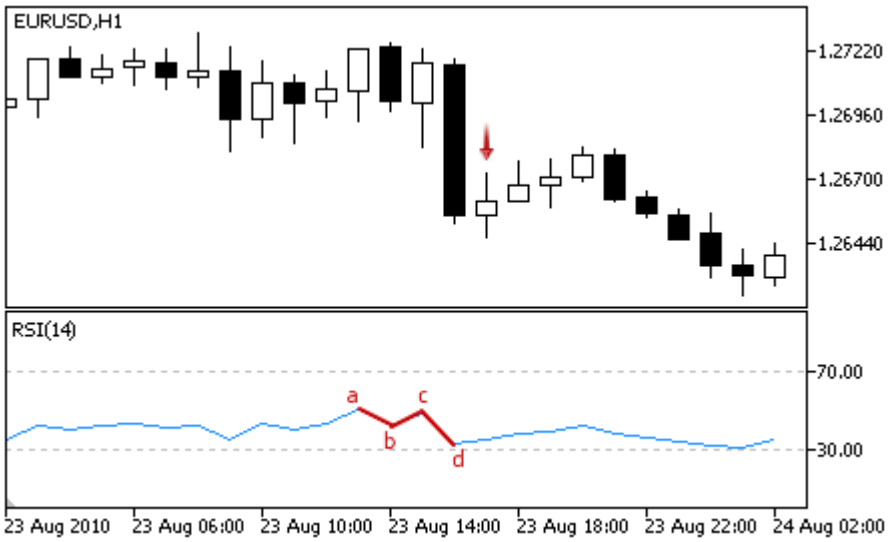
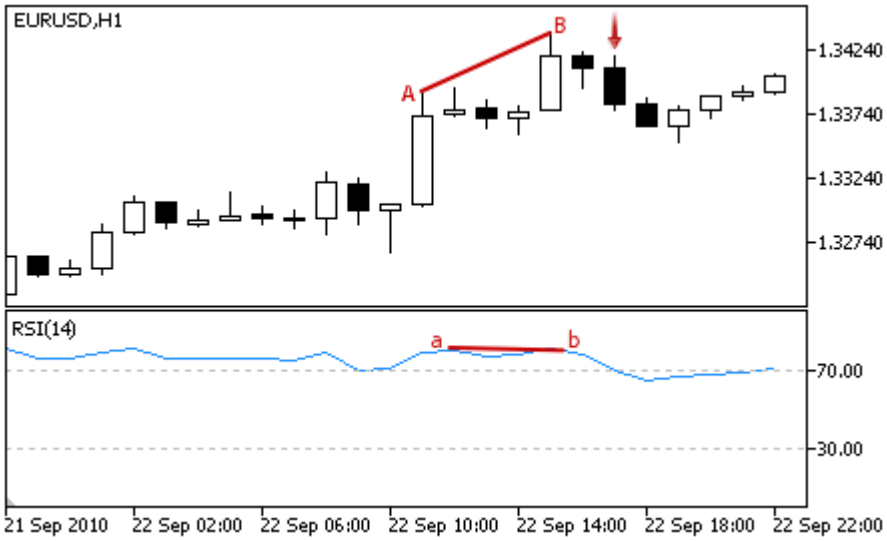
© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续底部，每个底部都高出前一个；并且价格形成三个相应的底部，每个底部都低于前一个。<br>  <br>• 头/肩 —振荡器形成三个连续底部，且中间一个低于两侧。 |

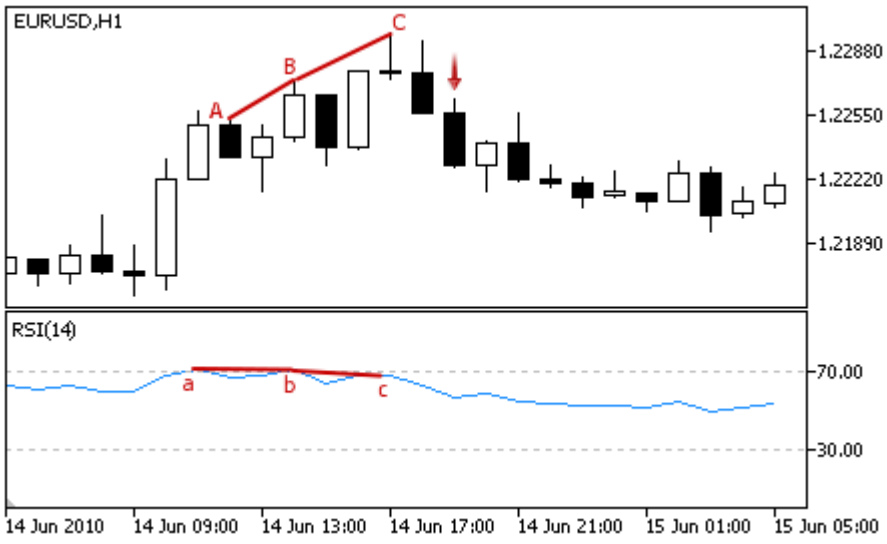
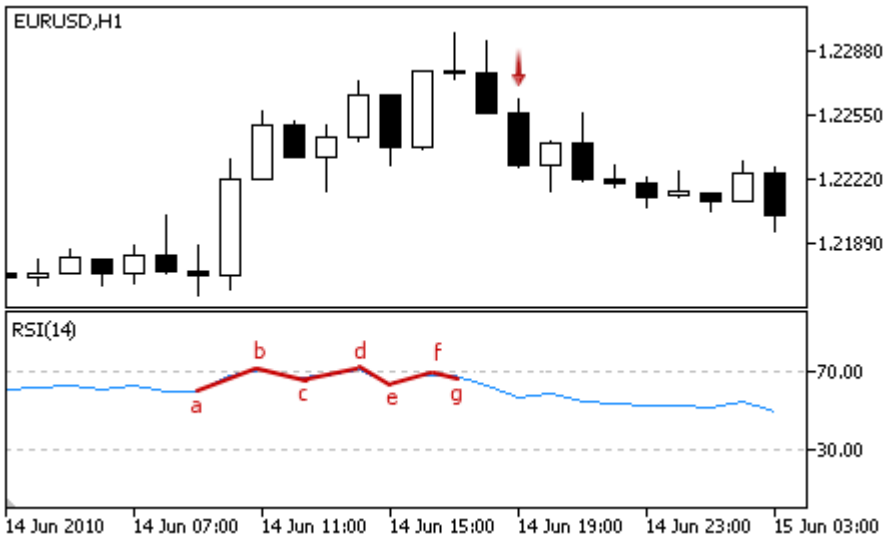
© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 超买之后的反转 —振荡器返身向下，而它在所分析的柱线上的值低于超买级别(省缺值是 70)。<br>  <br>• 波段失败 —振荡器值在所分析的柱线中下跌且低于前一底部。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 背离 — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。<br>  <br>• 双背离 —振荡器形成三个连续峰值，每个峰值都低于前一个；并且价格形成三个相应的峰值，每个峰值都高出前一个。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 头/肩 —振荡器形成三个连续峰值，且中间一个高于两侧。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodRSI | 振荡器计算的周期。 |
| Applied | 用于振荡器计算的 价格序列 。 |

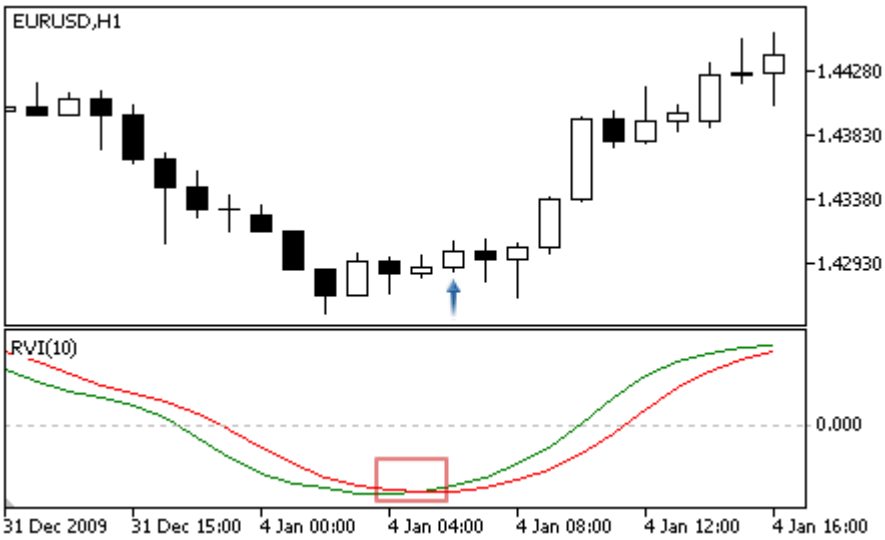
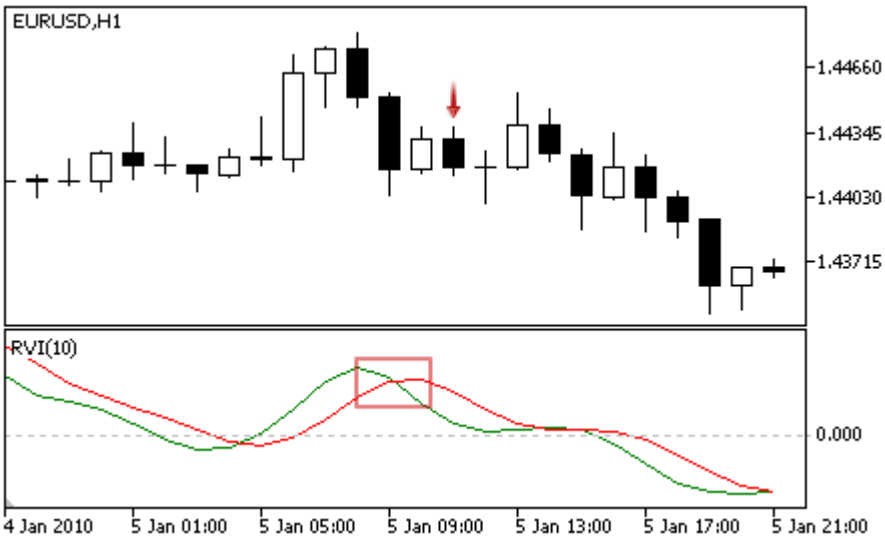
© 2000-2025, MetaQuotes Ltd.

# 相对动量指数振荡器信号

该信号模块基于 相对动量指数 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | **主线穿越信号线** —在所分析的柱线里，主线在信号线之上，且在前一根线里低于信号线。  |
| 对于做空 | **主线穿越信号线** —在所分析的柱线里，主线在信号线之下，且在前一根线里高于信号线。  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

© 2000-2025, MetaQuotes Ltd.

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| PeriodRVI | 振荡器计算的周期。 |

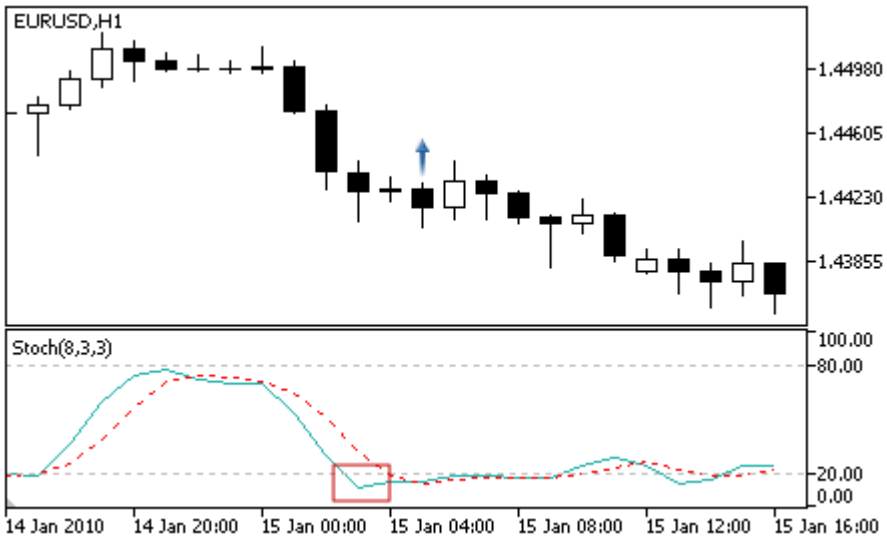
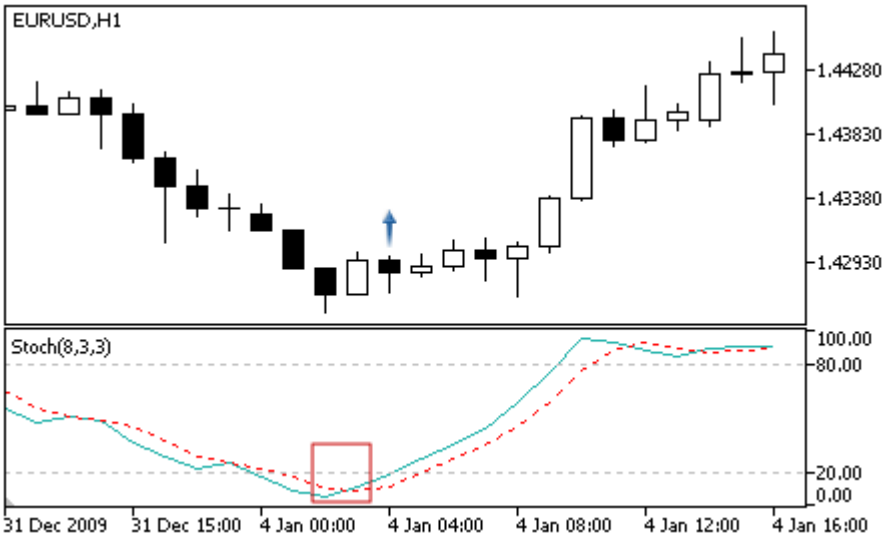
© 2000-2025, MetaQuotes Ltd.

# 随机振荡器信号

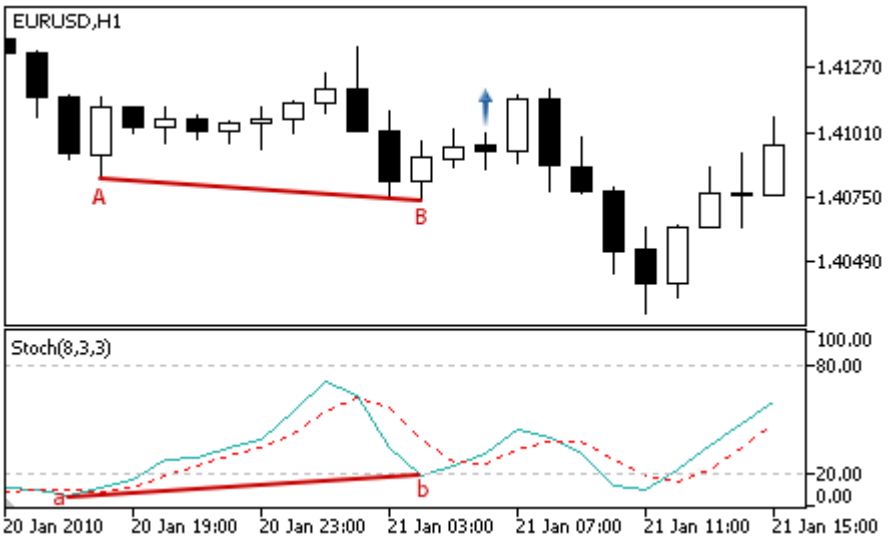
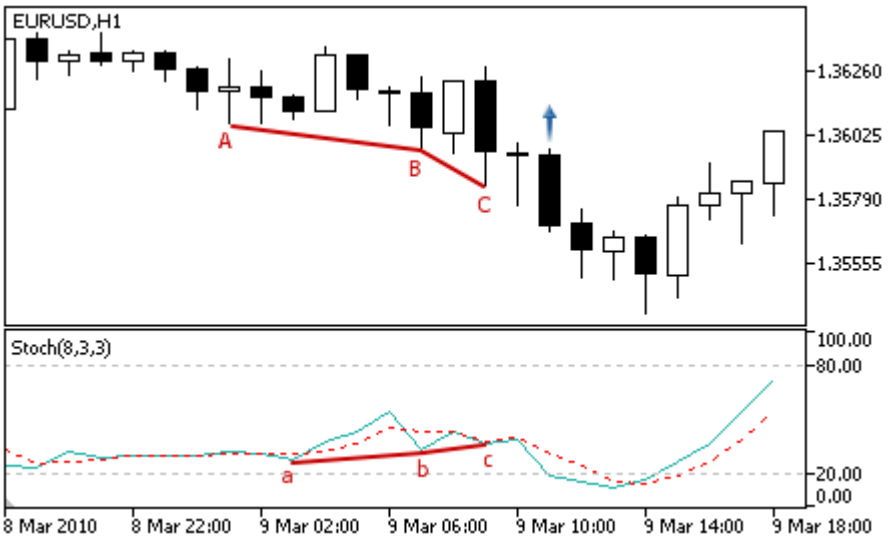
该信号模块基于 随机振荡器。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

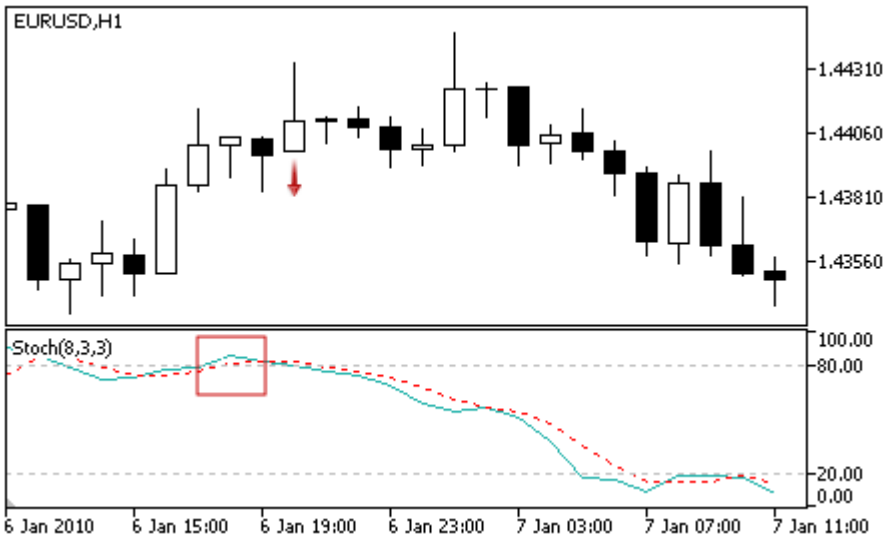
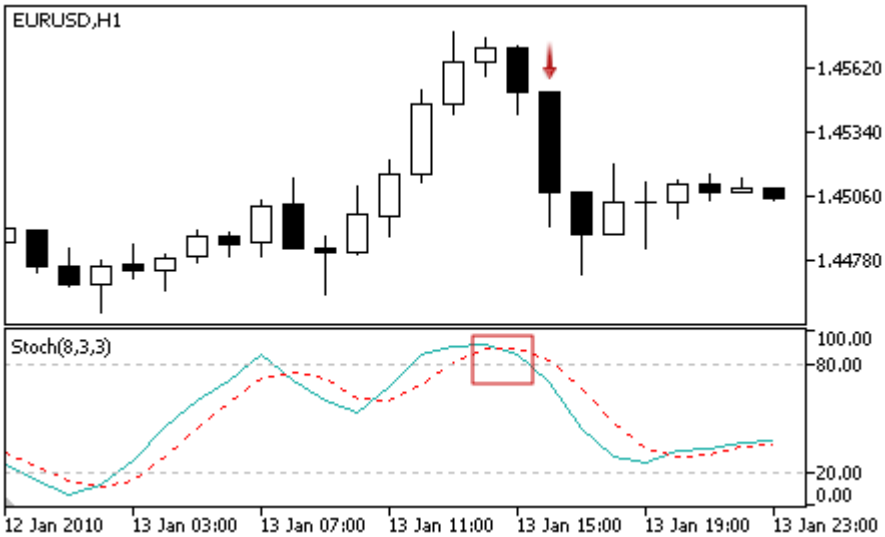
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **反转** —振荡器返身向上（振荡器在所分析的柱线里上升，而在前一根线里下降）。<br>  <br>• **主线穿越信号线** —在所分析的柱线里，主线在信号线之上，且在前一根线里低于信号线。<br>  <br>• **背离** — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。 |

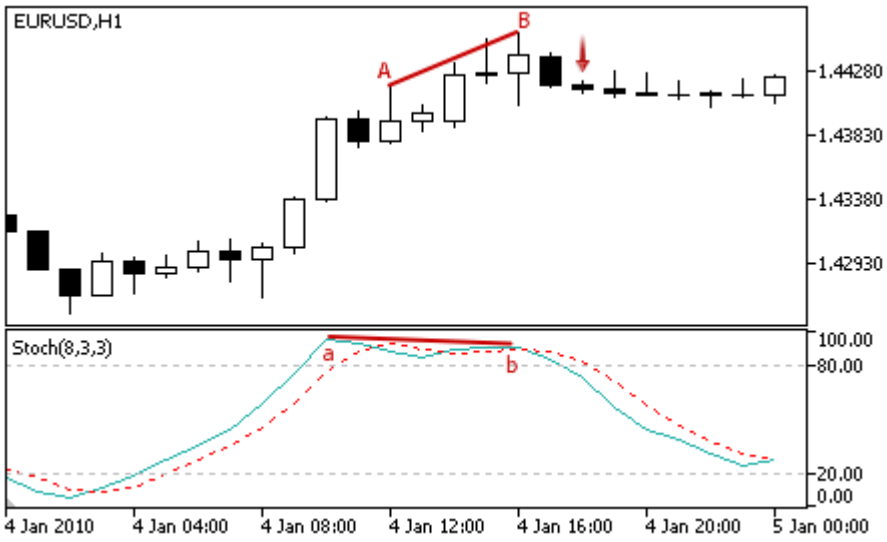
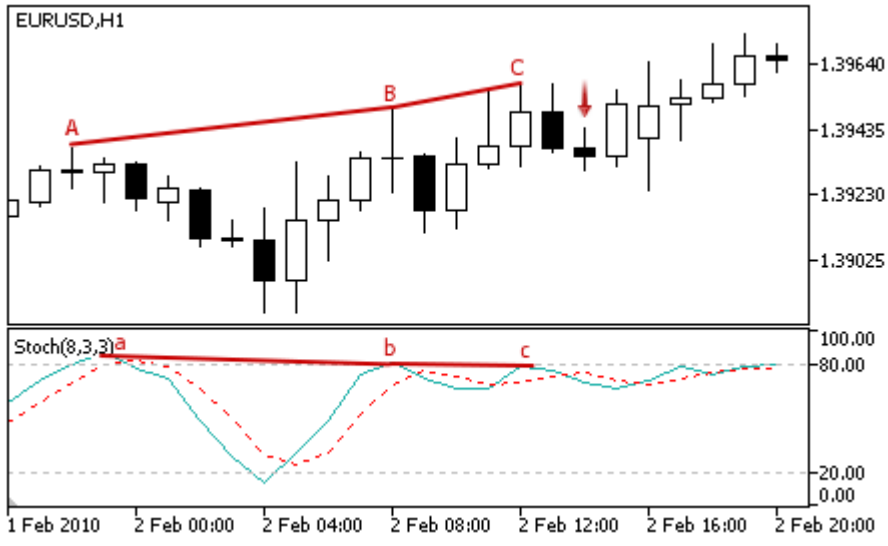
© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续底部，每个底部都高出前一个；并且价格形成三个相应的底部，每个底部都低于前一个。<br>  |
| 对于做空 | • 反转 —振荡器返身向下（振荡器在所分析的柱线里下降，而在前一根线里上升）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• **主线穿越信号线** — 在所分析的柱线里，主线在信号线之下，且在前一根线里高于信号线。<br>  <br>• **背离** — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 双背离 —振荡器形成三个连续峰值，每个峰值都低于前一个；并且价格形成三个相应的峰值，每个峰值都高出前一个。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodK | 振荡器主线的计算周期。 |
| PeriodD | 振荡器主线的计算周期。 |
| PeriodSlow | 慢速周期。 |
| Applied | 用于振荡器计算的 价格序列 。 |

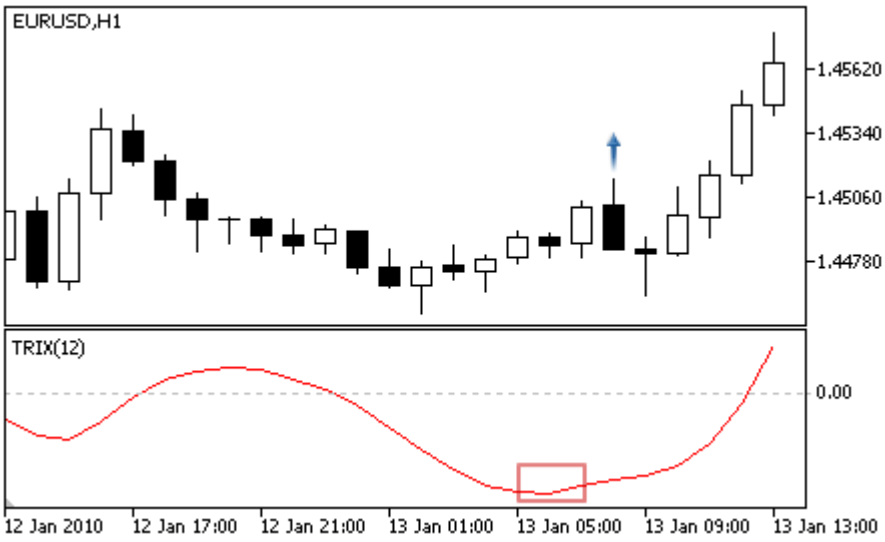
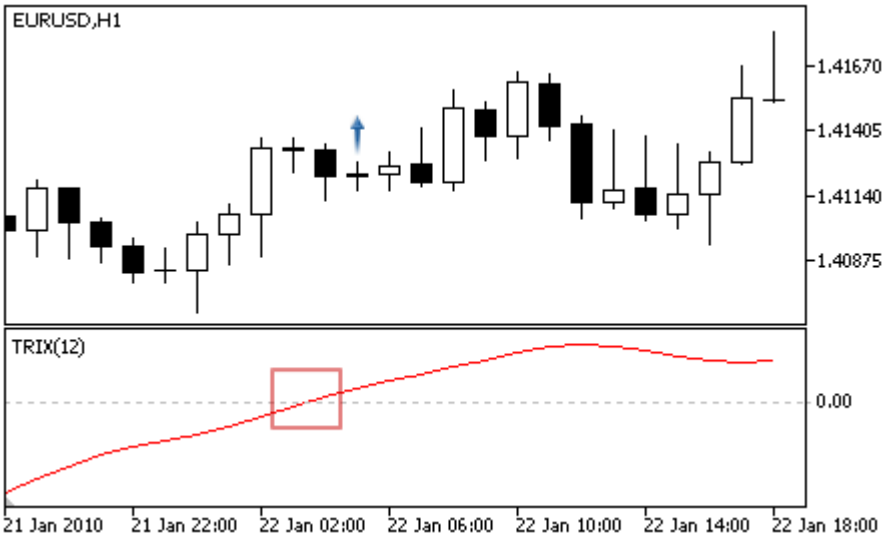
© 2000-2025, MetaQuotes Ltd.

# 三重指数均线振荡器信号

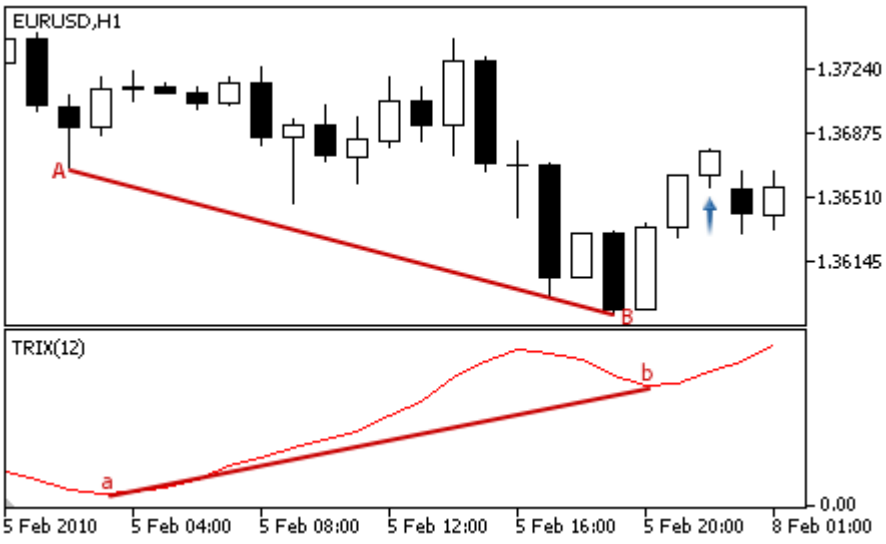
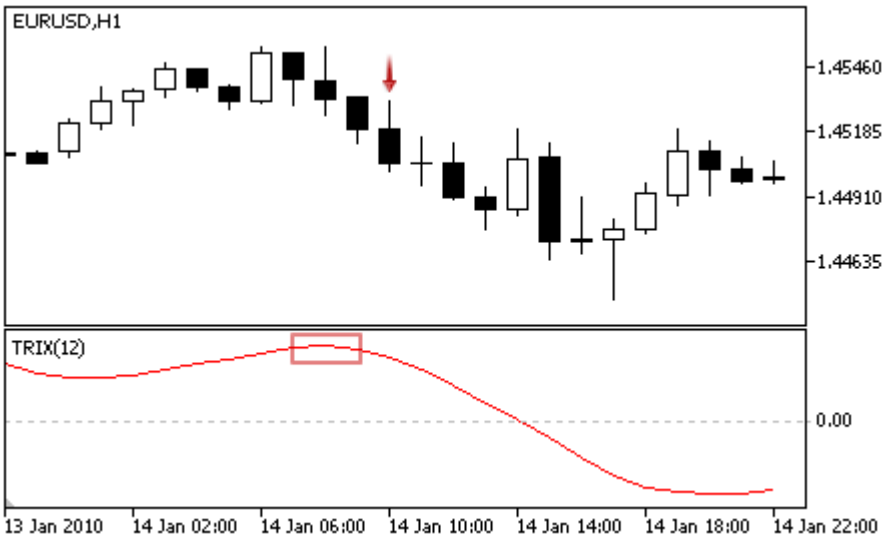
该信号模块基于 三重指数均线 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

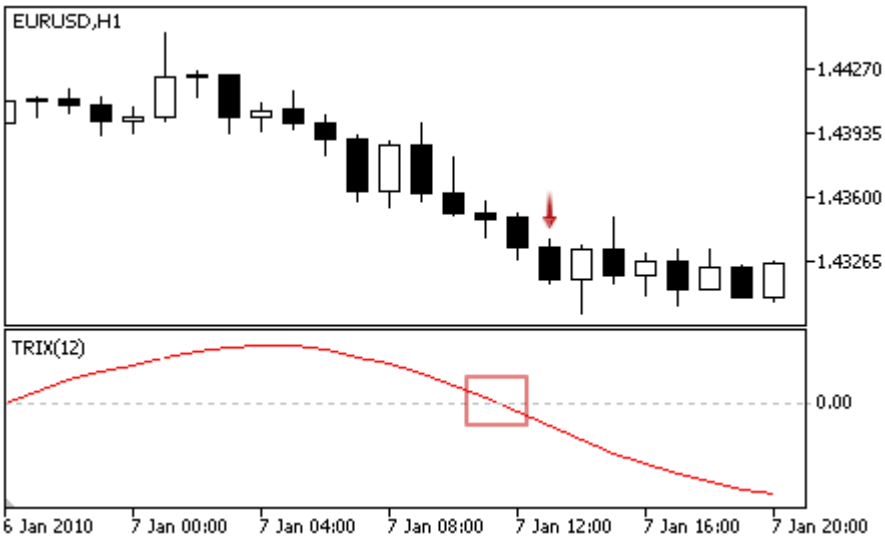
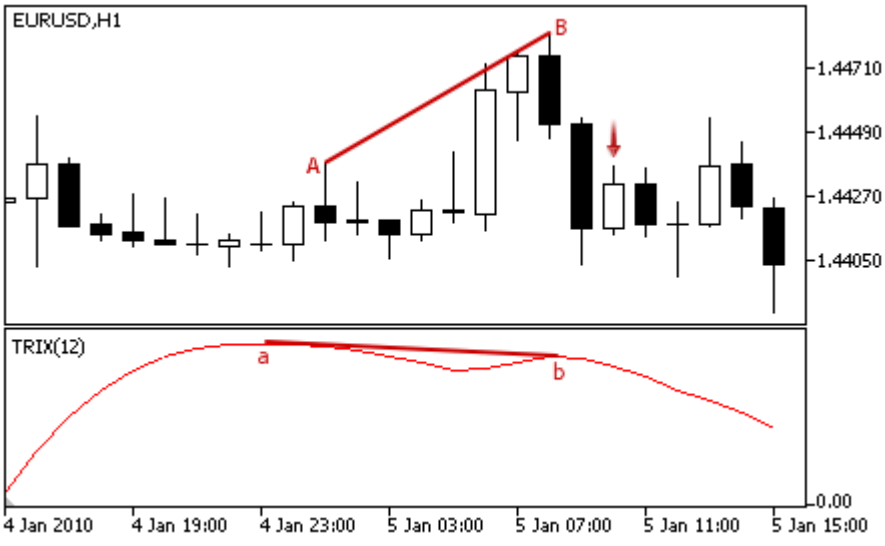
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **反转** —振荡器返身向上（振荡器在所分析的柱线里上升，而在前一根线里下降）。<br>  <br>• **穿越零轴** — 主线在所分析的柱线上高于零轴，且在前根柱线上低于零轴。<br>  <br>• **背离** — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 反转 —振荡器返身向下（振荡器在所分析的柱线里下降，而在前一根线里上升）。<br>  <br>• 穿越零轴 — 主线在所分析的柱线上低于零轴，且在前根柱线上高于零轴。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 背离 — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodTriX | 振荡器计算的周期。 |
| Applied | 用于振荡器计算的 价格序列 。 |

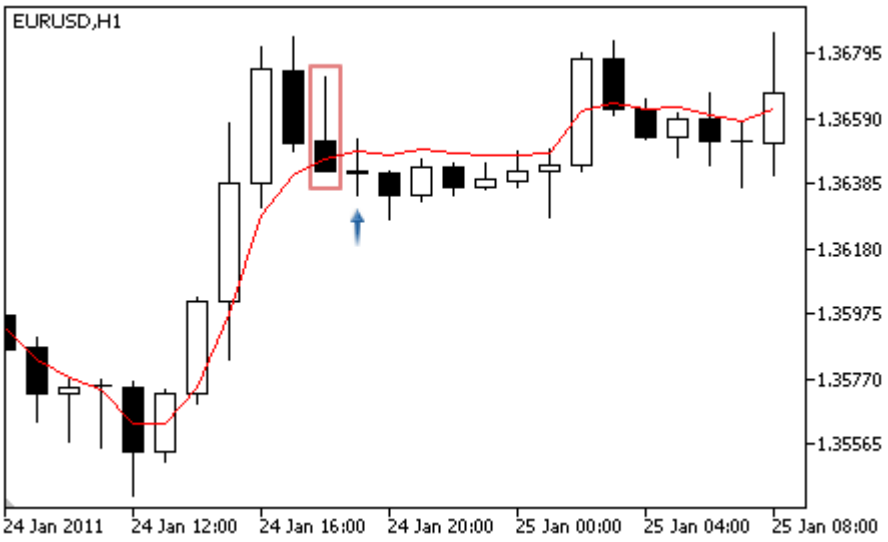
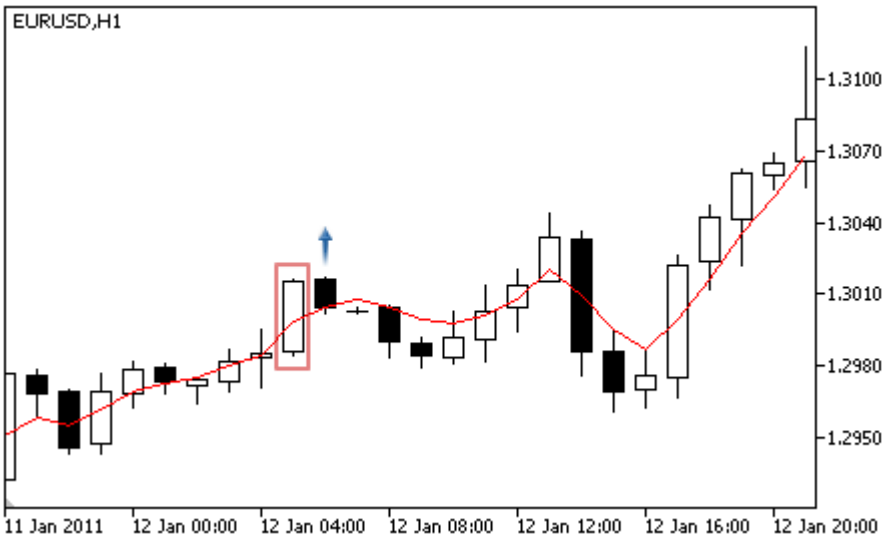
© 2000-2025, MetaQuotes Ltd.

# 三重指数移动均线指标信号

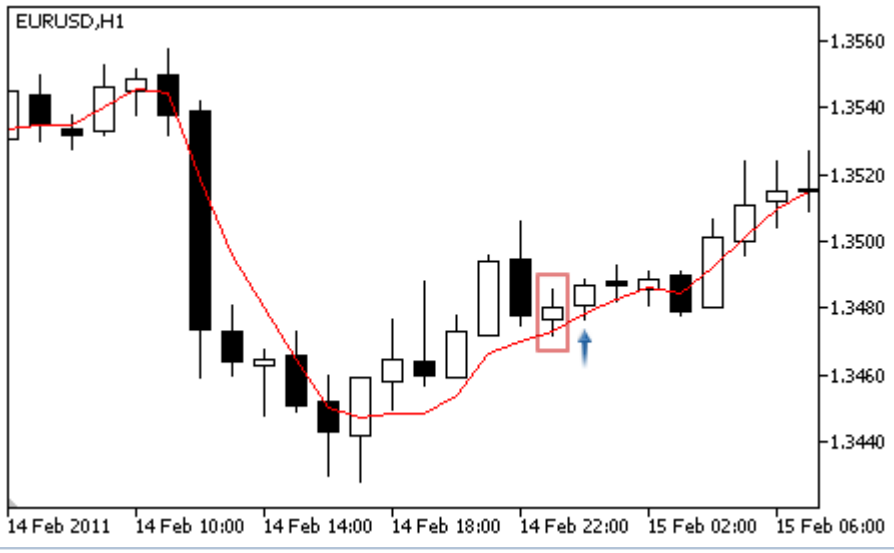
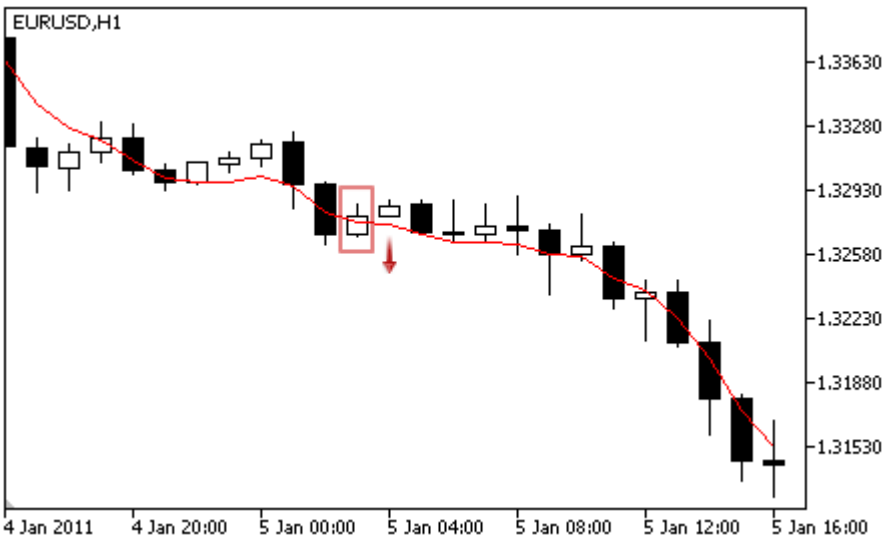
该信号模块基于 三重指数移动均线 指标的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

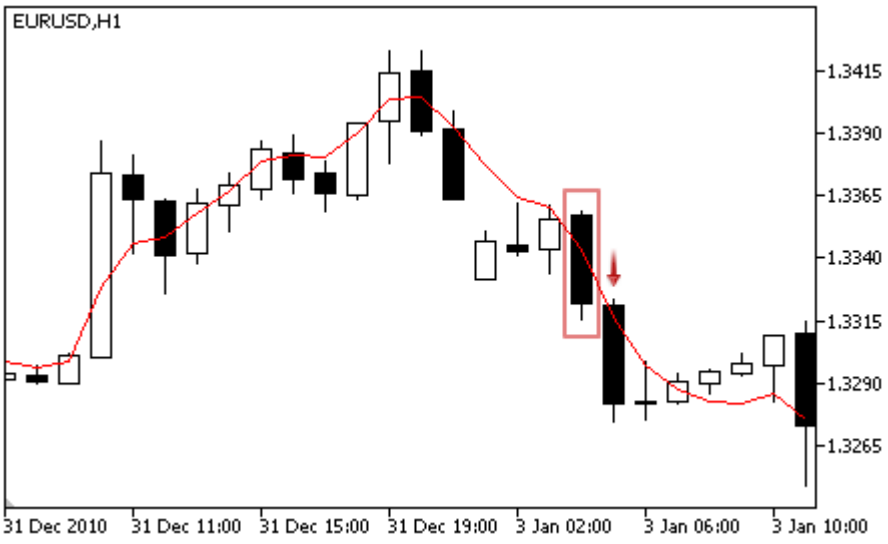
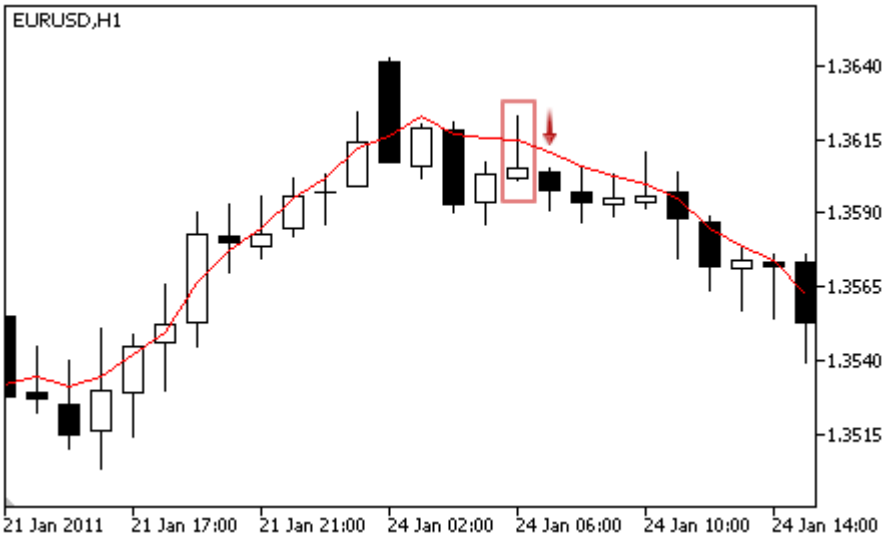
在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • 价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标上涨（弱信号）。<br>  <br>• **移动均线穿越**。价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标上涨（强信号）。<br>  <br>• 柱线的下影线穿越指标（所分析柱线的开盘价和收盘价均高于指标，而最低价低于指标）且指标上涨（弱信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  |
| 对于做空 | • 价格上穿指标（所分析柱线的开盘价低于指标，而收盘价高于指标）且指标下跌（弱信号）。<br>  <br>• **移动均线穿越**。价格下穿指标（所分析柱线的开盘价高于指标，而收盘价低于指标）且指标下跌（强信号）。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 柱线的上影线穿越指标（所分析柱线的开盘价和收盘价均低于指标，而最高价高于指标）且指标下跌（弱信号）。<br>  |
| 不反对做多 | 价格在指标之上。 |
| 不反对做空 | 价格在指标之下。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

## 调整参数

该模块有以下可调整参数：

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| PeriodMA | 均线指标的周期。 |
| 平移 | 指标相对于时间轴的平移（单位：柱线）。 |
| Method | 平均方法。 |
| Applied | 用于指标计算的 价格序列 。 |

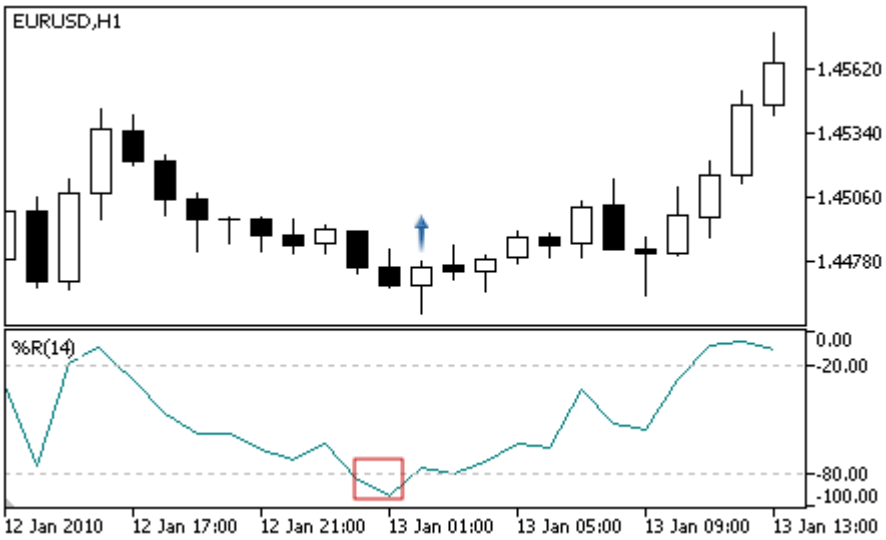
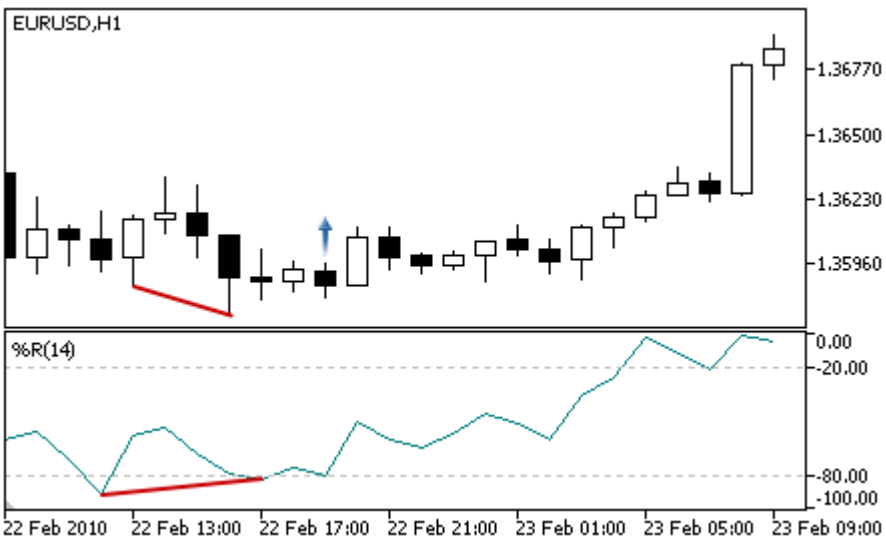
© 2000-2025, MetaQuotes Ltd.

# 威廉姆斯百分比范围振荡器信号

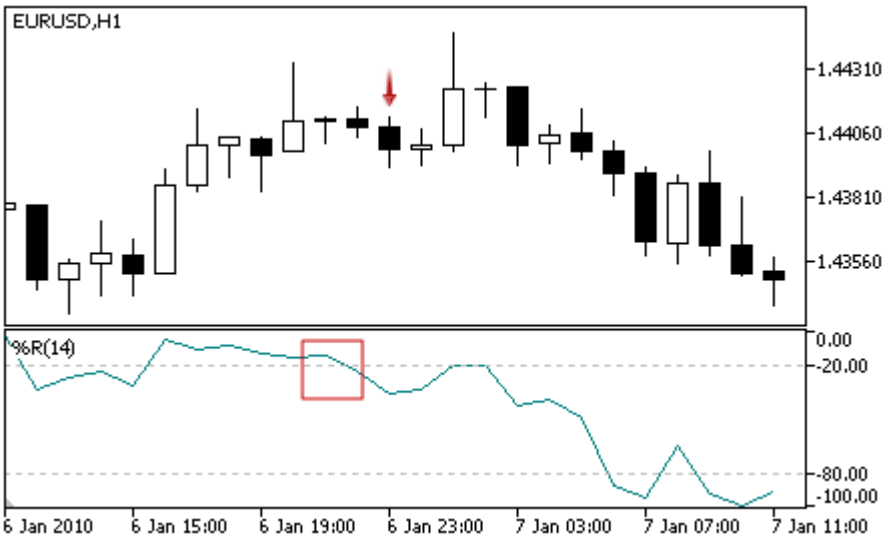
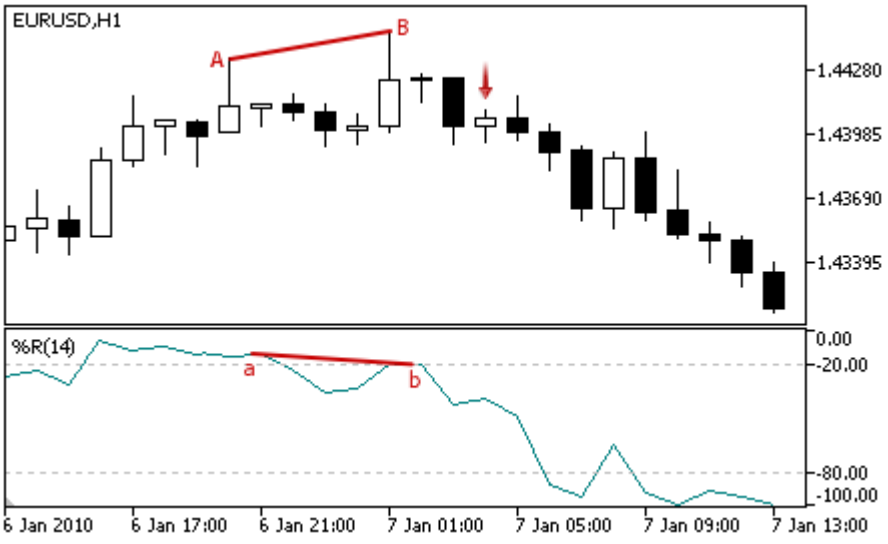
该信号模块基于 威廉姆斯百分比范围 振荡器的市场模型。关于从模块那里获取信号并制订交易决策的机制在 单独章节 里描述。

## 信号生成的条件

在以下部分，您可以发现有关模块传递信号到智能交易的条件描述。

| 信号类型 | 条件描述 |
| --- | --- |
| 对于做多 | • **超卖之后的反转** —振荡器返身向上，而它在所分析的柱线上的值高于超卖级别(省缺值是 -20)。  • **背离** — 振荡器最近分析的底部比更早的一个要高，而相应的价格波谷比更早的一个要低。  |
| 对于做空 | • **超买之后的反转** —振荡器返身向下，而它在所分析的柱线上的值低于超买级别(省缺值是 -80)。 |

© 2000-2025, MetaQuotes Ltd.

| 信号类型 | 条件描述 |
| --- | --- |
| |  <br>• 背离 — 振荡器最近分析的峰值比更早的一个要低，而相应的价格峰值比更早的一个要高。<br>  |
| 不反对做多 | 振荡器值在所分析的柱线中上涨。 |
| 不反对做空 | 振荡器值在所分析的柱线中下跌。 |

注释

依据智能交易的操作模式（"每笔报价"或"只做开盘价"）所分析的柱线即可是当前柱线（索引为 0），也可是最后一根形成的柱线（索引为 1）。

记住，威廉姆斯百分比范围振荡器有一个反向尺度。它的最大值是 -100，最小值是 0。

## 调整参数

该模块有以下可调整参数：

© 2000-2025, MetaQuotes Ltd.

| 参数 | 描述 |
| --- | --- |
| 权值 | 模块信号权值的间隔为 0 至 1。 |
| PeriodWPR | 振荡器计算的周期。 |

© 2000-2025, MetaQuotes Ltd.

# 尾随停止类

这章节包含操作尾随停止类的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建（和测试）交易策略时，使用这些类，将节省大量时间。

MQL5 标准库（就交易策略而言）位于终端工作目录的 Include\Expert\Trailing 文件夹。

| 类 | 描述 |
| --- | --- |
| CTrailingFixedPips | 该类实现的尾随停止算法，基于固定点数 |
| CTrailingMA | 该类实现的尾随停止算法，基于移动均线指标值 |
| CTrailingNone | 一个存根类，它不使用任何尾随停止算法 |
| CTrailingPSAR | 该类实现的尾随停止算法，基于抛物线转向指标值 |

© 2000-2025, MetaQuotes Ltd.

# CTrailingFixedPips

CTrailingFixedPips 类实现尾随停止算法，基于固定点数尾随。

如果仓位有止损价，它检查距现价的最小允许止损价。如果其值较低，则建议将止损级别设置到新的价位。对于有止赢价的仓位，则建议设置到新的止赢价。

如果智能交易 初始化 时标志 every_tick=false，它将仅在信柱线生成时执行所有操作（交易，尾随，等等）。此种情况下可以使用止盈级别。它可令您在新柱线完成之前将已开仓位在止盈价平仓。

## 描述

CTrailingFixedPips 实现的尾随停止算法，基于按照固定点数进行仓位尾随。

## 声明

```
class CTrailingFixedPips: public CExpertTrailing
```

## 标称库文件

```
   #include <Expert\Trailing\CTrailingFixedPips.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertTrailing
      CTrailingFixedPips

## 类方法

| 初始化 | |
|---|---|
| StopLevel | 设置止损级别的值。 |
| ProfitLevel | 设置止盈级别的值。 |
| virtual ValidationSettings | 验证设置 |
| 检查尾随方法 | |
| virtual CheckTrailingStopLong | 检查多头仓位的尾随停止条件 |
| virtual CheckTrailingStopShort | 检查空头仓位的尾随停止条件 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

© 2000-2025, MetaQuotes Ltd.

# StopLevel

设置止损级别的值（以点数）。

```
void  StopLevel(
   int  stop_level      // 止损级别
   )
```

参数

*stop_loss*

[输入] 止损级别的新值（以常规的 2/4-小数位）。

注释

如果止损级别等于 0，则不使用尾随停止。

© 2000-2025, MetaQuotes Ltd.

# ProfitLevel

设置止盈级别的值（以点数）。

```
void  ProfitLevel(
   int  profit_level      // 止盈级别
   )
```

参数

*profit_level*

[输入] 止盈级别的新值（以常规的 2/4-小数位）。

注释

如果止盈级别等于 0，则不使用尾随停止。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

函数检查止盈和止损级别。正确的值是 0，以及大于该品种的最小停止值的数值。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查多头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // CPositionInfo 对象指针
   double&         sl,            // 止损价位
   double&         tp             // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

如果止损级别等于 0，则不使用尾随停止。如果仓位有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果当前卖出价高于基准价+止损级别，则建议设置新的止损价。在此情况，如果仓位已有止盈价，则建议设置新的止盈价等于卖出价+止盈级别。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查空头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo 对象指针
   double&         sl,           // 止损价位
   double&         tp            // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

如果止损级别等于 0，则不使用尾随停止。如果仓位有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果当前买入价低于基准价-止损级别，则建议设置新的止损价。在此情况，如果仓位已有止盈价，则建议设置新的止盈价等于买入价-止盈级别。

© 2000-2025, MetaQuotes Ltd.

# CTrailingMA

CTrailingMA 类实现的尾随停止算法，基于移动均线指标值。

## 描述

CTrailingMA 类实现的尾随停止算法，基于前一根（完成）柱线的移动均线指标值。

## 声明

```
class CTrailingMA: public CExpertTrailing
```

## 标称库文件

```
   #include <Expert\Trailing\TrailingMA.mqh>
```

## 继承体系

COBject
  CExpertBase
    CExpertTrailing
      CTrailingMA

## 类方法

| 初始化 | |
|---|---|
| Period | 设置移动均线周期 |
| 平移 | 设置移动均线平移 |
| Method | 设置移动均线平滑方法 |
| Applied | 设置移动均线应用价格 |
| virtual InitIndicators | 初始化指标和时间序列 |
| virtual ValidationSettings | 验证设置 |
| 检查尾随方法 | |
| virtual CheckTrailingStopLong | 检查多头仓位的尾随停止条件 |
| virtual CheckTrailingStopShort | 检查空头仓位的尾随停止条件 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

© 2000-2025, MetaQuotes Ltd.

# Period

设置移动均线周期。

```
void  Period(
   int  period      // 平滑周期
   )
```

参数

*period*

[输入]移动均线周期。

© 2000-2025, MetaQuotes Ltd.

# 平移

设置移动均线平移。

```
void  Shift(
   int  shift      // 平移
   )
```

参数

*shift*

[输入]移动均线平移。

© 2000-2025, MetaQuotes Ltd.

# Method

设置移动均线平滑方法。

```
void  Method(
   ENUM_MA_METHOD  method      // 平滑方法
   )
```

参数

*method*

[输入] 移动均线指标的 平滑方法。

© 2000-2025, MetaQuotes Ltd.

# Applied

设置移动均线应用价格。

```
void  Applied(
   ENUM_APPLIED_PRICE  applied      // 应用价格
   )
```

参数

*applied*

[输入] MA 的 应用价格。

© 2000-2025, MetaQuotes Ltd.

# InitIndicators

初始化指标和时间序列。

```
virtual bool  InitIndicators(
   CIndicators*  indicators      // CIndicators 指针集合
   )
```

参数

*indicators*

[输入] 指标和时间序列指针的集合 (CExpert 类方法)。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

函数检查移动均线周期，正确值是正数。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查多头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // CPositionInfo 对象指针
   double&         sl,            // 止损价位
   double&         tp             // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

首先它计算最大允许的止损价，利用前一根（完成）柱线的移动均线指标值来计算最接近现价的止损价。

如果仓位已有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果计算出的止损价高于基准价且低于最大允许止损价，则建议设置新的止损价。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查空头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo 对象指针
   double&         sl,           // 止损价位
   double&         tp            // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

首先它计算最小允许的止损价，利用前一根（完成）柱线的移动均线指标值来计算最接近现价的止损价。

如果仓位已有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果计算出的止损价高于基准价且低于最小允许止损价，则建议设置新的止损价。

© 2000-2025, MetaQuotes Ltd.

# CTrailingNone

CTrailingNone 是一个存根 类。如果您的策略不使用尾随停止，则应使用这个类来初始化尾随对象。

## 描述

CTrailingNone 类未实现任何尾随停止算法。检查尾随停止条件的方法永远返回 false。

## 声明

```
class CTrailingNone: public CExpertTrailing
```

## 标称库文件

```
   #include <Expert\Trailing\TrailingNone.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertTrailing
      CTrailingNone

## 类方法

| 检查尾随方法 | |
|---|---|
| virtual CheckTrailingStopLong | 检查多头仓位尾随停止条件的存根方法 |
| virtual CheckTrailingStopShort | 检查空头仓位尾随停止条件的存根方法 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类 CExpertTrailing

CheckTrailingStopLong, CheckTrailingStopShort

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查多头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // CPositionInfo 对象指针
   double&         sl,            // 止损价位
   double&         tp             // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

该函数永远返回 false。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查空头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo 对象指针
   double&         sl,           // 止损价位
   double&         tp            // 止盈价位
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

该函数永远返回 false。

© 2000-2025, MetaQuotes Ltd.

# CTrailingPSAR

CTrailingPSAR 类实现的尾随停止算法，基于抛物线转向指标值。

## 描述

CTrailingPSAR 类实现的尾随停止算法，基于前一根（完成）柱线的抛物线转向指标值。

## 声明

```
class CTrailingPSAR: public CExpertTrailing
```

## 标称库文件

```
   #include <Expert\Trailing\TrailingParabolicSAR.mqh>
```

## 继承体系

COBject
  CExpertBase
    CExpertTrailing
      CTrailingPSAR

## 类方法

| 初始化 | |
|---|---|
| Step | 设置抛物线转向指标的步长值。 |
| Maximum | 设置抛物线转向指标的最大值。 |
| virtual InitIndicators | 初始化指标和时间序列 |
| 检查尾随方法 | |
| virtual CheckTrailingStopLong | 检查多头仓位的尾随停止条件 |
| virtual CheckTrailingStopShort | 检查空头仓位的尾随停止条件 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries

© 2000-2025, MetaQuotes Ltd.

## Step

设置抛物线转向指标的步长值。

```
void  Step(
   double  step      // 步长
   )
```

参数

*step*

[输入]抛物线转向指标的步长值。

© 2000-2025, MetaQuotes Ltd.

# Maximum

设置抛物线转向指标的最大值。

```
void  Maximum(
   double  maximum      // 最大值
   )
```

参数

*maximum*

[输入]抛物线转向指标的最大值。

© 2000-2025, MetaQuotes Ltd.

# InitIndicators

初始化指标和时间序列。

```
virtual bool  InitIndicators(
   CIndicators*  indicators      // CIndicators 指针集合
   )
```

参数

*indicators*

[输入] 指标和时间序列指针的集合 (CExpert 类方法)。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopLong

检查多头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // 指针
   double&         sl,            // 链表
   double&         tp             // 链表
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

首先它计算最大允许的止损价，利用前一根（完成）柱线的抛物线转向指标值来计算最接近现价的止损价。

如果仓位有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果计算出的止损价高于基准价且低于最大允许止损价，则建议设置新的止损价。

© 2000-2025, MetaQuotes Ltd.

# CheckTrailingStopShort

检查空头仓位的尾随停止条件。

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,      // 指针
   double&         sl,            // 链表
   double&         tp             // 链表
   )
```

参数

*position*

[输入] 类CPositionInfo对象的指针。

*sl*

[输入][输出] 止损价变量。

*tp*

[输入][输出] 止赢价变量。

返回值

true 如果条件满足，否则 false。

注释

首先它计算最小允许的止损价，利用前一根（完成）柱线的抛物线转向指标值来计算最接近现价的止损价。

如果仓位有止损价，这个值假定为基准价，否则以仓位的开仓价假定为基准价。

如果计算出的止损价高于基准价且低于最小允许止损价，则建议设置新的止损价。

© 2000-2025, MetaQuotes Ltd.

# 资金管理类

这章节包含操作资金和风险管理类的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建（和测试）交易策略时，使用这些类，将节省大量时间。

MQL5 标准库（就资金和风险管理类而言）位于终端工作目录的 Include\Expert\Money\ 文件夹。

| 类 | 描述 |
| --- | --- |
| CMoneyFixedLot | 该类实现的资金管理算法，基于预定义固定手数进行交易。 |
| CMoneyFixedMargin | 该类实现的资金管理算法，基于预定义固定保证金进行交易。 |
| CMoneyFixedRisk | 该类实现的资金管理算法，基于预定义风险进行交易。 |
| CMoneyNone | 该类实现的资金管理算法，基于最小手数进行交易。 |
| CMoneySizeOptimized | 该类实现的资金管理算法，基于变化手数进行交易，变化手数依据之前成交的结果。 |

© 2000-2025, MetaQuotes Ltd.

# CMoneyFixedLot

CMoneyFixedLot类是资金管理算法，基于预定义固定手数进行交易。

## 描述

CMoneyFixedLot实现的资金管理算法，基于预定义固定手数进行交易。

## 声明

```
class CMoneyFixedLot: public CExpertMoney
```

## 标称库文件

```
#include <Expert\Money\MoneyFixedLot.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertMoney
      CMoneyFixedLot

## 类方法

| 初始化 | |
|---|---|
| Lots | 设置交易量 |
| virtual ValidationSettings | 验证设置 |
| 资金和风险管理方法 | |
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类CExpertMoney

Percent, CheckReverse, CheckClose

© 2000-2025, MetaQuotes Ltd.

# Lots

设置交易量(单位:手数)。

```
void  Lots(
   double  lots      // 手数
   )
```

参数

*lots*

[输入] 交易量(单位:手数)。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

检查指定交易量的正确性。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

该函数永远返回由 Lots 方法定义的固定交易量。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取空头仓位的交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

空头仓位的交易量。

注释

该函数永远返回由 Lots 方法定义的固定交易量。

© 2000-2025, MetaQuotes Ltd.

# CMoneyFixedMargin

CMoneyFixedMargin 类是资金管理算法，基于预定义固定保证金进行交易。

## 描述

CMoneyFixedMargin 实现的资金管理算法，基于预定义固定保证金进行交易。

## 声明

```
class CMoneyFixedMargin: public CExpertMoney
```

## 标称库文件

```
#include <Expert\Money\MoneyFixedMargin.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertMoney
      CMoneyFixedMargin

## 类方法

| 资金和风险管理方法 | |
|---|---|
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类 CExpertMoney

Percent, ValidationSettings, CheckReverse, CheckClose

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

该函数返回多头仓位的交易量，它使用固定保证金。保证金由 CExpertMoney 基准类的百分比参数定义。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取空头仓位交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

空头仓位的交易量。

注释

该函数返回空头仓位的交易量，它使用固定保证金。保证金由 CExpertMoney 基准类的百分比参数定义。

© 2000-2025, MetaQuotes Ltd.

# CMoneyFixedRisk

CMoneyFixedRisk 类实现的资金管理算法基于固定的预定义风险。

## 描述

CMoneyFixedRisk 类实现的资金管理算法基于固定的预定义风险。

## 声明

```
class CMoneyFixedRisk: public CExpertMoney
```

## 标称库文件

```
#include <Expert\Money\MoneyFixedRisk.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertMoney
      CMoneyFixedRisk

## 类方法

| 资金和风险管理方法 | |
|---|---|
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类 CExpertMoney

Percent, ValidationSettings, CheckReverse

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

函数返回多头仓位的交易量，它使用固定风险。风险由 CExpertMoney 基准类的百分比参数定义。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取空头仓位交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

空头仓位的交易量。

注释

函数返回空头仓位的交易量，它使用固定风险。风险由 CExpertMoney 基准类的百分比参数定义。

© 2000-2025, MetaQuotes Ltd.

# CMoneyNone

CMoneyNone 类实现的交易算法基于最小允许手数。

## 描述

CMoneyNone 类实现的交易算法基于最小允许手数。

## 声明

```
class CMoneyNone: public CExpertMoney
```

## 标称库文件

```
#include <Expert\Money\MoneyNone.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertMoney
      CMoneyNone

## 类方法

| 初始化 | |
|---|---|
| virtual ValidationSettings | 验证设置 |
| 资金和风险管理方法 | |
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |

方法继承自类 CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类 CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类 CExpertMoney

Percent, CheckReverse, CheckClose

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

该函数永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

该函数永远返回最小手数。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取多头仓位的交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

空头仓位的交易量。

注释

该函数永远返回最小手数。

© 2000-2025, MetaQuotes Ltd.

# CMoneySizeOptimized

CMoneySizeOptimized 类实现的资金管理算法，基于变化手数进行交易，变化手数依据之前成交的结果。

## 描述

CMoneySizeOptimized 类实现的资金管理算法，基于变化手数进行交易，变化手数依据之前成交的结果。

## 声明

```
class CMoneySizeOptimized: public CExpertMoney
```

## 标称库文件

```
#include <Expert\Money\MoneySizeOptimized.mqh>
```

## 继承体系

CObject
  CExpertBase
    CExpertMoney
      CMoneySizeOptimized

## 类方法

| 初始化 | |
|---|---|
| DecreaseFactor | 设置递减因子值 |
| virtual ValidationSettings | 验证设置 |
| 资金和风险管理方法 | |
| virtual CheckOpenLong | 获取多头仓位的交易量 |
| virtual CheckOpenShort | 获取空头仓位的交易量 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

方法继承自类CExpertMoney

Percent, CheckReverse, CheckClose

© 2000-2025, MetaQuotes Ltd.

# DecreaseFactor

设置递减因子值。

```
void  DecreaseFactor(
   double  decrease_factor      // 递减因子
   )
```

参数

*decrease_factor*

[输入] 递减因子。

注释

当遭遇连续亏损时，递减因子定义交易量递减系数（与之前仓位的交易量相比）。

© 2000-2025, MetaQuotes Ltd.

# ValidationSettings

验证设置。

```
virtual bool  ValidationSettings()
```

返回值

true 如果成功，否则 false。

注释

如果递减因子值为负数，它返回 false，否则返回 true。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenLong

获取多头仓位的交易量。

```
virtual double  CheckOpenLong(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

该函数返回多头交易量，交易量依据之前成交的结果。

© 2000-2025, MetaQuotes Ltd.

# CheckOpenShort

获取空头仓位的交易量。

```
virtual double  CheckOpenShort(
   double  price,     // 价位
   double  sl         // 止损价位
   )
```

参数

*price*

[输入] 价位。

*sl*

[输入] 止损价位。

返回值

多头仓位的交易量。

注释

该函数返回空头交易量，交易量依据之前成交的结果。

© 2000-2025, MetaQuotes Ltd.

# 创建控制面板和对话框的类

这章节包含操作创建控制面板类的技术细节，以及 MQL5 标准库中相关组件的描述。

在创建 MQL5 的控制面板程序（脚本，智能交易）时，使用这些类，将节省大量时间。

MQL5 标准库（就控制面板而言）位于终端工作目录的 MQL5\Include\Controls 文件夹。

在以下文章中找到使用类的示例：

- 如何创建任何复杂等级的图形面板
- 改善面板：添加透明度、更改背景色和承自CAppDialog/CWndClient
- 立即向指标或EA交易添加控制面板
- 在MQL5中创建您自己的图形面板
- 在MQL5中创建用于交易的活动控制面板

在 MQL5\Expert\Examples\Controls 文件夹里，可找到描绘这些类是如何工作的智能交易例程。

| 辅助结构 | 描述 |
|---|---|
| CRect | 长方形区域结构 |
| CDateTime | 操作日期和时间的结构 |

| 基准类 | 描述 |
|---|---|
| CWnd | 所有控件的基准类 |
| CWndObj | 控件和对话框基准类 |
| CWndContainer | 复合控件基准类（含依赖控件） |

| 简单控件 Simple controls | 描述 |
|---|---|
| CLabel | 控件，基于 "文本标签" 图形对象 |
| CBmpButton | 控件，基于 "位图标签" 图形对象 |
| CButton | 控件，基于 "按钮" 图形对象 |
| CEdit | 控件，基于 "编辑域" 图形对象 |
| CPanel | 控件，基于 "长方形标签" |
| CPicture | 控件，基于 "位图标签" |

| 复合控件 | 描述 |
|---|---|
| CScroll | 滚动条基准类 |
| CScrollV | 垂直滚动条 |
| CScrollH | 水平滚动条 |

© 2000-2025, MetaQuotes Ltd.

| 复合控件 | 描述 |
| --- | --- |
| CWndClient | 客户区域滚动条基准类 |
| CListView | 列表视野 |
| CComboBox | 下拉框 |
| CCheckBox | 勾选框 |
| CCheckGroup | 勾选组 |
| CRadioButton | 单选框 |
| CRadioGroup | 单选组 |
| CSpinEdit | 旋转编辑框 |
| CDialog | 对话框 |
| CAppDialog | 应用对话框 |

© 2000-2025, MetaQuotes Ltd.

# CRect

CRect是图表上的长方形区域类。

## 描述

CRect区域类，它是在笛卡尔坐标系中由左上角和右下角坐标定义的一个长方形。

## 声明

```
class CRect
```

## 标称库文件

```
#include <Controls\Rect.mqh>
```

## 类方法

| 属性 | |
|---|---|
| Left | 获取/设置左上角X坐标 |
| Top | 获取/设置左上角Y坐标 |
| Right | 获取/设置右下角X坐标 |
| Bottom | 获取/设置右下角Y坐标 |
| Width | 获取/设置宽度 |
| Height | 获取/设置高度 |
| SetBound | 设置CRect类使用的新坐标 |
| Move | 设置CRect类的新坐标 |
| 平移 | 执行CRect坐标的相对平移 |
| Contains | 检查画点是否在CRect类区域内部 |
| 附加方法 | |
| Format | 获取区域坐标的字符串描述 |

© 2000-2025, MetaQuotes Ltd.

## Left (Get 方法)

获取/设置左上角 X 坐标。

```
int  Left()
```

返回值

左上角 X 坐标。

## Left (Set 方法)

设置左上角 X 坐标。

```
void  Left(
   const int  x      // 新的 x 坐标
   )
```

参数

*x*

[输入] 左上角的新 X 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Top (Get方法)

获取左上角 Y 坐标。

```
int  Top()
```

返回值

左上角 Y 坐标。。

## Top (Set方法)

设置左上角 Y 坐标。

```
void  Top(
   const int  y      // y 坐标
   )
```

参数

*y*
[输入] 新的 左上角 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Right (Get 方法)

获取右下角 X 坐标。

```
int  Right()
```

返回值

右下角 X 坐标。

## Right (Set 方法)

设置右下角 Y 坐标。

```
void  Right(
   const int  x      // x 坐标
   )
```

参数

*x*

[输入] 右下角的新 X 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Bottom (Get方法)

获取右下角Y坐标。

```
int  Bottom()
```

返回值

右下角Y坐标。

## Bottom (Set方法)

设置右下角Y坐标。

```
void  Bottom(
   const int  y      // y 坐标
   )
```

参数

*y*

[输入] 新的右下角Y坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Width (Get方法)

获取区域宽度。

```
int  Width()
```

返回值

区域宽度。

## Width (Set方法)

设置区域新宽度。

```
virtual bool  Width(
   const int  w      // 宽度
   )
```

参数

*w*

[输入] 新宽度。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Height (Get 方法)

获取区域高度。

```
int  Height()
```

返回值

区域高度。

## Height (Set 方法)

设置区域新高度。

```
virtual bool  Height(
   const int  h      // 高度
   )
```

参数

*h*

[输入] 新高度。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## SetBound

设置使用 CRect 类坐标的新坐标。

```
void  SetBound(
   const & CRect  rect      // CRect 类
   )
```

返回值

无。

## SetBound

设置区域新坐标。

```
void  SetBound(
   const int  l      // 左
   const int  t      // 顶
   const int  r      // 右
   const int  b      // 底
   )
```

参数

*l*

[输入] 左上角的新 X 坐标。

*t*

[输入] 左上角的新 Y 坐标。

*r*

[输入] 右下角 X 坐标。

*b*

[输入] 右下角 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Move

设置 CRect 类的新坐标。

```
void  Move(
   const int  x,      // X 坐标
   const int  y       // Y 坐标
   )
```

参数

*x*

[输入] 新的 X 坐标。

*y*

[输入] 新的 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Shift

执行 CRect 类坐标的相对平移。

```
void  Shift(
   const int  dx,      // 增量 X
   const int  dy       // 增量 Y
   )
```

参数

*dx*

[输入] 增量 X。

*dy*

[输入] 增量 Y。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Contains

检查画点是否在 CRect 类区域内部。

```
bool  Contains(
   const int  x,      // X 坐标
   const int  y       // Y 坐标
   )
```

参数

*x*

[输入] X 坐标。

*y*

[输入] Y 坐标。

返回值

true，如果画点在区域内部（包括边框），否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Format

获取区域坐标的字符串描述。

```
string  Format(
   string &  fmt,     // 格式
   ) const
```

参数

*fmt*

[输入] 字符串格式。

返回值

区域坐标的字符串。

© 2000-2025, MetaQuotes Ltd.

# CDateTime

CDateTime 是操作日期和时间的结构。

## 描述

CDateTime 是由 MqlDateTime 派生出的结构，它用来操作控件的日期和时间。

## 声明

```
struct CDateTime
```

## 标称库文件

```
#include <Tools\DateTime.mqh>
```

## 类方法

| 属性 | |
|---|---|
| MonthName | 获取月份名 |
| ShortMonthName | 获取月份名简称 |
| DayName | 获取周内日期名 |
| ShortDayName | 获取周内日期名简称 |
| DaysInMonth | 获取月内天数 |
| Get/Set 方法 | |
| DateTime | 获取/设置日期和时间 |
| Date | 设置日期 |
| Time | 设置时间 |
| Sec | 设置秒数 |
| Min | 设置分钟数 |
| Hour | 设置小时数 |
| Day | 设置月内天数 |
| Mon | 设置月份 |
| Year | 设置年份 |
| 附加方法 | |
| SecDec | 相减指定秒数 |
| SecInc | 相加指定秒数 |
| MinDec | 相减指定分钟数 |

© 2000-2025, MetaQuotes Ltd.

| 属性 | |
|---|---|
| MinInc | 相加指定分钟数 |
| HourDec | 相减指定小时数 |
| HourInc | 相加指定小时数 |
| DayDec | 相减指定天数 |
| DayInc | 相加指定天数 |
| MonDec | 相减指定月数 |
| MonInc | 相加指定月数 |
| YearDec | 相减指定年数 |
| YearInc | 相加指定年数 |

© 2000-2025, MetaQuotes Ltd.

# MonthName

获取月份名。

```
string  MonthName()  const
```

通过指数获得月份名称。

```
string  MonthName(
   const int       num          // 月份索引
   )  const
```

参数

*num*

[输入] 月份索引 (1-12)。

返回值

月份名。

© 2000-2025, MetaQuotes Ltd.

# ShortMonthName

获取月份名简称。

```
string  ShortMonthName()  const
```

获取索引处的月份名简称。

```
string  ShortMonthName(
   const int       num          // 月份索引
   )  const
```

参数

*num*

[输入] 月份索引 (1-12)。

返回值

月份名简称。<

© 2000-2025, MetaQuotes Ltd.

# DayName

获取周内日期名。

```
string  DayName()  const
```

获取索引处的周内日期名。

```
string  DayName(
   const int       num          // 日期索引
   )  const
```

参数

*num*

[输入] 日期索引 (0-6)。

返回值

日期名。

© 2000-2025, MetaQuotes Ltd.

# ShortDayName

获取周内日期名简称。

```
string  ShortDayName()  const
```

获取索引处的周内日期名。

```
string  ShortDayName(
   const int       num          // 日期索引
   )  const
```

参数

*num*

[输入] 日期索引 (0-6)。

返回值

日期名简称。

© 2000-2025, MetaQuotes Ltd.

# DaysInMonth

获取月内天数。

```
int  DaysInMonth()  const
```

返回值

月内天数。

© 2000-2025, MetaQuotes Ltd.

## DateTime (Get方法)

获取日期和时间。

```
datetime  DateTime()
```

返回值

值为 datetime 类型。

## DateTime (Set方法 datetime)

根据 datetime 类型设置日期和时间。

```
void  DateTime(
   const datetime      value          // 日期和时间
   )
```

参数

*value*

[输入] datetime 类型的值。

返回值

无。

## DateTime (Set方法 MqlDateTime)

根据 MqlDateTime 类型设置日期和时间。

```
void  DateTime(
   const MqlDateTime   &value         // 日期和时间
   )
```

参数

*value*

[输入] MqlDateTime 类型的值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Date (Set方法 datetime)

根据 datetime 类型设置日期。

```
void  Date(
   const datetime      value          // 日期
   )
```

参数

*value*

[输入] datetime 类型的值。

返回值

无。

## Date (Set方法 MqlDateTime)

根据 MqlDateTime 类型设置日期。

```
void  Date(
   const MqlDateTime   &value         // 日期
   )
```

参数

*value*

[输入] MqlDateTime 类型的值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Time (Set方法 datetime)

根据 datetime 类型设置时间。

```
void  Time(
   const datetime      value          // 时间
   )
```

参数

*value*

[输入] datetime 类型的值。

返回值

无。

## Time (Set方法 MqlDateTime)

根据 MqlDateTime 类型设置时间。

```
void  Time(
   const MqlDateTime   &value         // 时间
   )
```

参数

*value*

[输入] MqlDateTime 类型的值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Sec

设置秒数。

```
void  Sec(
   const int    value         // 秒数
   )
```

参数

*value*

[输入] 秒数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Min

设置分钟数。

```
void  Min(
   const int    value         // 分钟数
   )
```

参数

*value*

[输入] 分钟数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Hour

设置小时数。

```
void  Hour(
   const int    value         // 小时数
   )
```

参数

*value*

[输入] 小时数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Day

设置月内天数。

```
void  Day(
   const int    value         // 天数
   )
```

参数

*value*

[输入] 月内天数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Mon

设置月份。

```
void  Mon(
   const int    value        // 月份
   )
```

参数

*value*

[输入] 月份。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Year

设置年份。

```
void  Year(
   const int    value         // 年份
   )
```

参数

*value*

[输入] 年份。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## SecDec

相减指定秒数。

```
void  SecDec(
   int   delta=1        // 秒数
   )
```

参数

*delta*

[输入] 相减的秒数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# SecInc

相加指定秒数。

```
void  SecInc(
   int   delta=1        // 秒数
   )
```

参数

*delta*

[输入] 相加的秒数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# MinDec

相减指定分钟数。

```
void  MinDec(
   int   delta=1        // 分钟数
   )
```

参数

*delta*

[输入] 相减的分钟数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# MinInc

相加指定分钟数。

```
void  MinInc(
   int   delta=1        // 分钟数
   )
```

参数

*delta*

[输入] 相加的分钟数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# HourDec

相减指定小时数。

```
void  HourDec(
   int   delta=1        // 小时数
   )
```

参数

*delta*

[输入] 相减的小时数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# HourInc

相加指定小时数。

```
void  HourInc(
   int   delta=1        // 小时数
   )
```

参数

*delta*

[输入] 相加的小时数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DayDec

相减指定天数。

```
void  DayDec(
   int   delta=1        // 天数
   )
```

参数

*delta*

[输入] 相减的天数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# DayInc

相加指定天数。

```
void  DayInc(
   int   delta=1        // 天数
   )
```

参数

*delta*

[输入] 相加的天数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# MonDec

相减指定月数。

```
void  MonDec(
   int   delta=1        // 月数
   )
```

参数

*delta*

[输入] 相减的月数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# MonInc

相加指定月数。

```
void  MonInc(
   int   delta=1        // 月数
   )
```

参数

*delta*

[输入] 相加的月数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# YearDec

相减指定年数。

```
void  YearDec(
   int   delta=1        // 年数
   )
```

参数

*delta*

[输入] 相减的年数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# YearInc

相加指定年数。

```
void  YearInc(
   int   delta=1        // 年数
   )
```

参数

*delta*

[输入] 相加的年数。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CWnd

CWnd是包含在MQL5标准库中的所有控件的基准类。

## 描述

CWnd类是控件基准类的实现。

## 声明

```
class CWnd : public CObject
```

## 标称库文件

```
#include <Controls\Wnd.mqh>
```

## 继承体系

CObject

CWnd

直接分支

CDragWnd, CWndContainer, CWndObj

## 类方法

| | |
|---|---|
| 创建于销毁 | |
| 创建 | 创建控件 |
| 销毁 | 销毁控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| OnMouseEvent | 对应 CHARTEVENT_MOUSE_MOVE 事件的事件处理器 |
| 名称 | |
| 名称 | 获取控件名称 |
| 访问容器 | |
| ControlsTotal | 获取容器中控件的数量 |
| Control | 获取索引处的控件 |
| ControlFind | 根据ID获取控件 |
| 几何 | |
| Rect | 获取CRect类对象的指针 |
| Left | 获取/设置左上角X坐标 |

© 2000-2025, MetaQuotes Ltd.

| 创建于销毁 | |
|---|---|
| Top | 获取/设置左上角 Y 坐标 |
| Right | 获取/设置右下角 X 坐标 |
| Bottom | 获取/设置右下角 Y 坐标 |
| Width | 获取/设置宽度 |
| Height | 获取/设置高度 |
| Move | 设置控件新坐标。 |
| 平移 | 执行控件坐标的相对平移 |
| 调整大小 | 设置控件的新宽度/高度。 |
| Contains | 检查画点/控件是否在控件区域内部 |
| **Align** | |
| Alignment | 设置控件对齐属性。 |
| Align | 执行控件对齐 |
| 标识 | |
| Id | 获取/设置控件 ID |
| 状态 | |
| IsEnabled | 获取控件是否启用的指示值 |
| Enable | 设置控件是否启用的指示值 |
| Disable | 禁用控件 |
| IsVisible | 检查可视性标志 |
| Visible | 检查可视性标志 |
| Show | 显示控件 |
| Hide | 隐藏控件 |
| IsActive | 检查控件激活状态 |
| Activate | 激活控件 |
| Deactivate | 解除控件激活 |
| 状态标志 | |
| StateFlags | 获取/设置控件状态标志 |
| StateFlagsSet | 设置控件状态标志 |
| StateFlagsReset | 重置控件状态标志 |
| 属性标志 | |

© 2000-2025, MetaQuotes Ltd.

| 创建于销毁 | |
|---|---|
| PropFlags | 获取/设置控件属性标志 |
| PropFlagsSet | 设置控件属性标志 |
| PropFlagsReset | 重置控件属性标志 |
| 鼠标操作 | |
| MouseX | 获取/设置鼠标 X 坐标 |
| MouseY | 获取/设置鼠标 Y 坐标 |
| MouseFlags | 获取/设置鼠标按钮状态 |
| MouseFocusKill | 清除鼠标焦点 |
| 内部事件处理器 | |
| OnCreate | "创建"事件处理器 |
| OnDestroy | "销毁"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnResize | "调整大小"事件处理器 |
| OnEnable | "启用"事件处理器 |
| OnDisable | "禁用"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnActivate | "激活"事件处理器 |
| OnDeactivate | "解除激活"事件处理器 |
| OnClick | "点击"事件处理器 |
| OnChange | "修改"事件处理器 |
| 鼠标事件处理器 | |
| OnMouseDown | "鼠标按下"事件处理器 |
| OnMouseUp | "鼠标抬起"事件处理器 |
| 拖动事件处理器 | |
| OnDragStart | "拖动开始"事件处理器 |
| OnDragProcess | "拖动过程"事件处理器 |
| OnDragEnd | "拖动结束"事件处理器 |
| 拖动对象 | |
| DragObjectCreate | 创建拖动对象 |

© 2000-2025, MetaQuotes Ltd.

| 创建于销毁 | |
|---|---|
| DragObjectDestroy | 销毁拖动对象 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

© 2000-2025, MetaQuotes Ltd.

# Create

创建控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的新 X 坐标。

*y1*

[输入] 左上角的新 Y 坐标。

*x2*

[输入] 右下角 X 坐标。

*y2*

[输入] 右下角 Y 坐标。

返回值

true 如果成功，否则 false。

注释

基准类中的方法只保存参数且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# Destroy

销毁控件。

```
virtual bool  Destroy()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMouseEvent

鼠标事件处理器（对应 CHARTEVENT_MOUSE_MOVE 图表事件）。

```
virtual bool  OnMouseEvent(
   const int  x,          // x 坐标
   const int  y,          // y 坐标
   const int  flags       // 标志
   )
```

参数

*x*

[输入] 鼠标光标相对于图表左上角的 X 坐标。

*y*

[输入] 鼠标光标相对于图表左上角的 Y 坐标。

*flags*

[输入] 鼠标按钮状态标志。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Name

获取控件名称。

```
string  Name()  const
```

返回值

控件名称。

© 2000-2025, MetaQuotes Ltd.

# ControlsTotal

获取容器中控件的数量。

```
int  ControlsTotal()  const
```

返回值

容器中控件的数量。

注释

基准类方法没有容器，它可供访问其继承者的容器并永远返回 0。

© 2000-2025, MetaQuotes Ltd.

# Control

获取索引处的控件。

```
CWnd*  Control(
   const int  ind      // 索引
   )  const
```

参数

*ind*

[输入] 控件索引。

返回值

控件指针。

注释

基准类方法没有容器，它可供访问其继承者的容器并永远返回 NULL。

© 2000-2025, MetaQuotes Ltd.

# ControlFind

从容器里获取指定 ID 的控件。

```
virtual CWnd*  ControlFind(
   const long  id      // ID
   )
```

参数

*id*

[输入] 用于搜索的控件标识符。

返回值

控件指针。

注释

基准类方法没有容器，它可供访问其继承者的容器。如果指定 ID 匹配容器 ID，它返回自身的指针 (this)。

© 2000-2025, MetaQuotes Ltd.

# Rect

获取 CRect 类对象的指针。

```
const CRect*  Rect()  const
```

返回值

CRect 类对象的指针。

© 2000-2025, MetaQuotes Ltd.

## Left (Get 方法)

获取控件左上角的 X 坐标。

```
int  Left()
```

返回值

控件左上角的 X 坐标。

## Left (Set 方法)

设置控件左上角的 X 坐标。

```
void  Left(
   const int  x      // 新的 x 坐标
   )
```

参数

*x*

[输入] 左上角的新 X 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Top (Get方法)

获取控件左上角的 Y 坐标。

```
int  Top()
```

返回值

控件左上角的 Y 坐标。

## Top (Set方法)

设置控件左上角的 Y 坐标。

```
void  Top(
   const int  y      // y 坐标
   )
```

参数

*y*

[输入] 新的 左上角 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Right (Get 方法)

获取控件右下角的 X 坐标。

```
int  Right()
```

返回值

右下角 X 坐标。

## Right (Set 方法)

设置控件右下角的 Y 坐标。

```
void  Right(
   const int  x      // x 坐标
   )
```

参数

*x*

[输入] 右下角的新 X 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Bottom (Get方法)

获取控件右下角的 Y 坐标。

```
int  Bottom()
```

返回值

控件右下角的 Y 坐标。

## Bottom (Set方法)

设置控件右下角的 Y 坐标。

```
void  Bottom(
   const int  y      // y 坐标
   )
```

参数

*y*

[输入] 新的右下角 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Width (Get方法)

获取控件宽度。

```
int  Width()
```

返回值

控制的宽度 。

## Width (Set方法)

设置控件的新宽度。

```
virtual bool  Width(
   const int  w      // 宽度
   )
```

参数

*w*

[输入] 新宽度。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Height (Get 方法)

获取控件高度。

```
int  Height()
```

返回值

控件高度。

## Height (Set 方法)

设置控件的新高度。

```
virtual bool  Height(
   const int  h      // 高度
   )
```

参数

*h*

[输入] 新高度。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Move

保存控件的新坐标。

```
void  Move(
   const int  x,      // X 坐标
   const int  y       // Y 坐标
   )
```

参数

*x*

[输入] 新的 X 坐标。

*y*

[输入] 新的 Y 坐标。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# 平移

执行控件坐标的相对平移。

```
void  Shift(
   const int  dx,      // 增量 X
   const int  dy       // 增量 Y
   )
```

参数

*dx*

[输入] 增量 X。

*dy*

[输入] 增量 Y。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# 调整大小

设置控件的新宽度/高度。

```
virtual bool  Resize(
   const int  w,     // 宽度
   const int  h      // 高度
   )
```

参数

*w*

[输入] 新宽度。

*h*

[输入] 新高度。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Contains

检查画点是否在图表的控件区域内部

```
bool  Contains(
   const int  x,      // X 坐标
   const int  y       // Y 坐标
   )
```

参数

*x*

[输入] X 坐标。

*y*

[输入] Y 坐标。

返回值

true，如果画点在区域内部（包括边框），否则 - false。

## Contains

检查指定控件是否在图表的控件区域内部。

```
bool  Contains(
   const CWnd*  control      // 指针
   )  const
```

参数

*control*

[输入] 对象指针。

返回值

true，如果指定控件在区域内部（包含边框），否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Alignment

设置控件对齐参数。

```
void  Alignment(
   const int  flags,       // 对齐标志
   const int  left,        // 自左边框的偏移
   const int  top,         // 自顶边框的偏移
   const int  right,       // 自右边框的偏移
   const int  bottom       // 自底边框的偏移
   )
```

参数

*flags*

[输入] 对齐标志。

*left*

[输入] 自左边框的固定偏移。

*top*

[输入] 自顶边框的固定偏移。

*right*

[输入] 自右边框的固定偏移。

*bottom*

[输入] 自底边框的固定偏移。

返回值

无。

注释

对齐标志：

```
enum WND_ALIGN_FLAGS
{
   WND_ALIGN_NONE=0,                                  // 无对齐
   WND_ALIGN_LEFT=1,                                  // 左对齐
   WND_ALIGN_TOP=2,                                   // 顶对齐
   WND_ALIGN_RIGHT=4,                                 // 右对齐
   WND_ALIGN_BOTTOM=8,                                // 底对齐
   WND_ALIGN_WIDTH = WND_ALIGN_LEFT|WND_ALIGN_RIGHT,  // 宽度对齐
   WND_ALIGN_HEIGHT=WND_ALIGN_TOP|WND_ALIGN_BOTTOM,   // 高度对齐
   WND_ALIGN_CLIENT=WND_ALIGN_WIDTH|WND_ALIGN_HEIGHT, // 高度和宽度对齐
   }
```

© 2000-2025, MetaQuotes Ltd.

# Align

在指定图表区域执行控件对齐。

```
virtual bool  Align(
   const CRect*  rect      // 指针
   )
```

参数

*rect*

[输入] 图表区域坐标处的对象指针。

返回值

true 如果成功，否则 false。

注释

对齐参数必须指定 (省缺不进行对齐)。

© 2000-2025, MetaQuotes Ltd.

## Id (Get方法)

获取控件 ID。

```
long  Id()  const
```

返回值

控件标识符。

## Id (Set方法)

设置控件的新 ID 值。

```
virtual long  Id(
   const long  id      // 标识符
   )
```

参数

*id*

[输入] 控件标识符的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# IsEnabled

获取控件是否启用的指示值。

```
bool  IsEnabled()  const
```

返回值

true - 如果控件启用，否则 - false。

© 2000-2025, MetaQuotes Ltd.

# Enable

启用控件。

```
virtual bool  Enable()
```

返回值

true 如果成功，否则 false。

注释

控件启用，它可处理外部事件。

© 2000-2025, MetaQuotes Ltd.

# Disable

禁用控件。

```
virtual bool  Disable()
```

返回值

true 如果成功，否则 false。

注释

被禁用的控件不能处理外部事件。

© 2000-2025, MetaQuotes Ltd.

# IsVisible

获取控件是否可视的指示值。

```
bool  IsVisible()  const
```

返回值

true 如果控件在图表上显示，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Visible

检查可视性标志。

```
virtual bool  Visible(
   const bool  flag      // 标志
   )
```

参数

*flag*

[输入] 新标志。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Show

显示控件。

```
virtual bool  Show()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Hide

隐藏控件。

```
virtual bool  Hide()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# IsActive

获取控件是否激活的指示值。

```
bool  IsActive()  const
```

返回值

true 如果控件激活，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Activate

激活控件。

```
virtual bool  Activate()
```

返回值

true 如果成功，否则 false。

注释

当鼠标光标悬浮其上时控件激活。

© 2000-2025, MetaQuotes Ltd.

# Deactivate

解除控件激活。

```
virtual bool  Deactivate()
```

返回值

true 如果成功，否则 false。

注释

当鼠标光标移出控件时控件处于未激活。

© 2000-2025, MetaQuotes Ltd.

## StateFlags (Get方法)

获取控件状态标志。

```
int  StateFlags()
```

返回值

控件状态标志。

## StateFlags (Set方法)

设置控件状态标志。

```
virtual void  StateFlags(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 新的控件状态标志。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# StateFlagsSet

设置控件状态标志。

```
virtual void  StateFlagsSet(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 设置的标志 (位掩模)。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# StateFlagsReset

重置控件状态标志。

```
virtual void  StateFlagsReset(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 重置的标志 (位掩模)。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## PropFlags (Get方法)

获取控件属性标志。

```
void  PropFlags(
   const int  flags      // 标志
   )
```

返回值

控件属性标志。

## PropFlags (Set方法)

设置控件属性标志。

```
virtual void  PropFlags(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 新标志。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# PropFlagsSet

设置控件属性标志。

```
virtual void  PropFlagsSet(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 设置的标志 (位掩模)。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# PropFlagsReset

重置控件属性标志。

```
virtual void  PropFlagsReset(
   const int  flags      // 标志
   )
```

参数

*flags*

[输入] 重置的标志 (位掩模)。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## MouseX (Set方法)

保存鼠标 X 坐标。

```
void  MouseX(
   const int  value      // 坐标
   )
```

参数

*value*

[输入] 鼠标的 X 坐标。

返回值

无。

## MouseX (Get方法)

获取保存的鼠标 X 坐标。

```
int  MouseX()
```

返回值

保存的鼠标 X 坐标。

© 2000-2025, MetaQuotes Ltd.

## MouseY (Set方法)

保存鼠标Y坐标。

```
void  MouseY(
   const int  value      // 坐标
   )
```

参数

*value*

[输入] 鼠标的Y坐标。

返回值

无。

## MouseY (Get方法)

获取保存的鼠标Y坐标。

```
int  MouseY()
```

返回值

保存的鼠标Y坐标。

© 2000-2025, MetaQuotes Ltd.

## MouseFlags (Set方法)

设置鼠标按钮的状态。

```
virtual void  MouseFlags(
   const int  value      // 状态
   )
```

参数

*value*

[输入] 鼠标按钮的状态。

返回值

无。

## MouseFlags (Get方法)

获取保存的鼠标按钮状态。

```
int  MouseFlags()
```

返回值

鼠标按钮状态。

© 2000-2025, MetaQuotes Ltd.

# MouseFocusKill

清除保存的鼠标按钮状态并解除控件激活。

```
bool  MouseFocusKill(
   const long  id=CONTROLS_INVALID_ID      // id
   )
```

参数

*id=CONTROLS_INVALID_ID*

[输入] 接收鼠标焦点的控件标识符。

返回值

控件解除激活结果。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnDestroy

控件的"销毁"事件处理器。

```
virtual bool  OnDestroy()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnEnable

控件的"齐用"(如果启用则能响应用户交互)事件处理器。

```
virtual bool  OnEnable()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnDisable

控件的"禁用"(如果禁用则不能响应用户交互)事件处理器。

```
virtual bool  OnDisable()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnActivate

控件的"激活"事件处理器。

```
virtual bool  OnActivate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnDeactivate

控件的"解除激活"事件处理器。

```
virtual bool  OnDeactivate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnClick

控件的"点击"(鼠标左键点击)事件处理器。

```
virtual bool  OnClick()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnMouseDown

控件的"鼠标按下"事件处理器。

```
virtual bool  OnMouseDown()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上按鼠标左键时"鼠标按下"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnMouseUp

控件的"鼠标抬起"事件处理器。

```
virtual bool  OnMouseUp()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上释放鼠标左键时"鼠标抬起"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnDragStart

控件的"拖动开始"事件处理器。

```
virtual bool  OnDragStart()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当开始拖动操作时"拖动开始"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnDragProcess

控件的"拖动过程"事件处理器。

```
virtual bool  OnDragProcess(
   const int  x,      // x 坐标
   const int  y       // y 坐标
   )
```

参数

*x*

[输入]当前鼠标光标的 X 坐标。

*y*

[输入] 当前鼠标光标的 Y 坐标。

返回值

true 如果事件已经被处理，否则 false。

注释

当控件移动时"拖动过程"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnDragEnd

控件的"拖动结束"事件处理器。

```
virtual bool  OnDragEnd()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当控件拖动完成"拖动结束"事件触发。

© 2000-2025, MetaQuotes Ltd.

# DragObjectCreate

创建拖动对象。

```
virtual bool  DragObjectCreate()
```

返回值

true 如果成功，否则 false。

注释

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# DragObjectDestroy

销毁拖动对象。

```
virtual bool  DragObjectDestroy()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CWndObj

CWndObj是标准库中简单控件（基于图表对象）的基准类。

## 描述

CWndObj类实现简单控件的基本方法。

## 声明

```
class CWndObj : public CWnd
```

## 标称库文件

```
#include <Controls\WndObj.mqh>
```

## 继承体系

CObject
  CWnd
    CWndObj

直接分支

CBmpButton, CButton, CEdit, CLabel, CPanel, CPicture

## 类方法

| 图表事件处理 | |
|---|---|
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| 文本 | 获取/设置 OBJPROP_TEXT 图表对象的属性 |
| Color | 获取/设置 OBJPROP_COLOR 图表对象的属性 |
| ColorBackground | 获取/设置 OBJPROP_BGCOLOR 图表对象的属性 |
| ColorBorder | 获取/设置 OBJPROP_BORDER_COLOR 图表对象的属性 |
| Font | 获取/设置 OBJPROP_FONT 图表对象的属性 |
| FontSize | 获取/设置 OBJPROP_FONTSIZE 图表对象的属性 |
| ZOrder | 获取/设置 OBJPROP_ZORDER 图表对象的属性 |
| 图表对象事件处理器 | |
| OnObjectCreate | CHARTEVENT_OBJECT_CREATE 事件处理器 |
| OnObjectChange | CHARTEVENT_OBJECT_CHANGE 事件处理器 |
| OnObjectDelete | CHARTEVENT_OBJECT_DELETE 事件处理器 |
| OnObjectDrag | CHARTEVENT_OBJECT_DRAG 事件处理器 |

© 2000-2025, MetaQuotes Ltd.

| 图表事件处理 | |
|---|---|
| 属性改变事件处理器 | |
| OnSetText | "设置文本"事件处理器 |
| OnSetColor | "设置颜色"事件处理器 |
| OnSetColorBackground | "设置背景颜色"事件处理器 |
| OnSetFont | "设置字体"事件处理器 |
| OnSetFontSize | "设置字号"事件处理器 |
| OnSetZOrder | "设置图层顺序"事件处理器 |
| 内部事件处理器 | |
| OnDestroy | "销毁"事件处理器 |
| OnChange | "修改"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Create, Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Text (Get方法)

获取 OBJPROP_TEXT (文本) 图表对象的属性。

```
string  Text()
```

返回值

对应的 OBJPROP_TEXT 属性值。

## Text (Set方法)

设置OBJPROP_TEXT (文本) 图表对象的属性。

```
bool  Text(
   const string  value      // 新值
   )
```

参数

*value*

[输入] 新的 OBJPROP_TEXT 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Color (Get方法)

获取 OBJPROP_COLOR (颜色) 图表对象的属性。

```
color  Color()
```

返回值

对应的 OBJPROP_COLOR 属性值。

## Color (Set方法)

设置 OBJPROP_COLOR (颜色) 图表对象的属性。

```
bool  Color(
   const color  value      // 值
   )
```

参数

*value*

[输入] 新的 OBJPROP_COLOR 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## ColorBackground (Get方法)

获取 OBJPROP_BGCOLOR (背景颜色) 图表对象的属性。

```
color  ColorBackground()
```

返回值

对应的 OBJPROP_BGCOLOR 属性值。

## ColorBackground (Set方法)

设置 OBJPROP_BGCOLOR (背景颜色) 图表对象的属性。

```
bool  ColorBackground(
   const color  value      // 值
   )
```

参数

*value*

[输入] 新的 OBJPROP_BGCOLOR 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## ColorBorder (Get方法)

获取 OBJPROP_BORDER_COLOR (边框颜色) 图表对象的属性。

```
color  ColorBorder()
```

返回值

对应的 OBJPROP_BORDER_COLOR 属性值。

## ColorBorder (Set方法)

设置 OBJPROP_BORDER_COLOR (边框颜色) 图表对象的属性。

```
bool  ColorBorder(
   const color  value      // 值
   )
```

参数

*value*

[输入] 新的 OBJPROP_BORDER_COLOR 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Font (Get方法)

获取 OBJPROP_FONT (字体) 图表对象的属性。

```
string  Font()
```

返回值

对应的 <li1>OBJPROP_FONT</li1><t2> 属性值</t2>。

## Font (Set方法)

设置 OBJPROP_FONT (字体) 图表对象的属性。

```
bool  Font(
   const string  value      // 新值
   )
```

参数

*value*

[输入] 新的 OBJPROP_FONT 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## FontSize (Get方法)

获取 OBJPROP_FONTSIZE (字号) 图表对象的属性。

```
int  FontSize()
```

返回值

对应的 OBJPROP_FONTSIZE 属性值。

## FontSize (Set方法)

设置 OBJPROP_FONTSIZE (字号) 图表对象的属性。

```
bool  FontSize(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的 OBJPROP_FONTSIZE 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## ZOrder (Get方法)

获取 OBJPROP_ZORDER 图表对象的属性。

```
long  ZOrder()
```

返回值

对应的 OBJPROP_ZORDER 属性值。

## ZOrder (Set方法)

设置 OBJPROP_ZORDER 图表对象的属性。

```
bool  ZOrder(
   const long  value      // 新值
   )
```

参数

*value*

[输入] 新的 OBJPROP_ZORDER 属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnObjectCreate

图表对象 CHARTEVENT_OBJECT_CREATE 事件处理器。

```
virtual bool  OnObjectCreate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnObjectChange

图表对象 CHARTEVENT_OBJECT_CHANGE 事件处理器。

```
virtual bool  OnObjectChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnObjectDelete

图表对象 CHARTEVENT_OBJECT_DELETE 事件处理器。

```
virtual bool  OnObjectDelete()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnObjectDrag

图表对象 CHARTEVENT_OBJECT_DRAG 事件处理器。

```
virtual bool  OnObjectDrag()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetText

控件的"设置文本"(改变 OBJPROP_TEXT 属性)事件处理器。

```
virtual bool  OnSetText()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnSetColor

控件的"设置颜色"(改变 OBJPROP_COLOR 属性)事件处理器。

```
virtual bool  OnSetColor()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBackground

控件的"设置背景颜色"(改变 OBJPROP_BGCOLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnSetFont

控件的"设置字体"(改变 OBJPROP_FONT 属性)事件处理器。

```
virtual bool  OnSetFont()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnSetFontSize

控件的"设置字号"(改变 OBJPROP_FONTSIZE 属性)事件处理器。

```
virtual bool  OnSetFontSize()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnSetZOrder

控件的"设置图层"(改变 OBJPROP_ZORDER 属性)事件处理器。

```
virtual bool  OnSetZOrder()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnDestroy

控件的"销毁"事件处理器。

```
virtual bool  OnDestroy()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CWndContainer

CWndContainer是标准库中复合控件的基准类。

## 描述

CWndContainer类实现复合控件的基本方法。

## 声明

```
class CWndContainer : public CWnd
```

## 标称库文件

```
#include <Controls\WndContainer.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer

直接分支

CCheckBox, CComboBox, CDateDropList, CDatePicker, CDialog, CRadioButton, CScroll, CSpinEdit, CWndClient

## 类方法

| | |
|---|---|
| 销毁 | |
| 销毁 | 销毁所有控件容器 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| OnMouseEvent | 响应 CHARTEVENT_MOUSE_MOVE 的事件处理器 |
| 访问容器 | |
| ControlsTotal | 获取容器中控件的数量 |
| Control | 获取索引处的控件 |
| ControlFind | 根据 ID 获取控件 |
| 添加/删除 | |
| 添加 | 添加控件至容器 |
| 删除 | 从容器中删除控件 |
| 几何 | |
| Move | 设置容器中所有控件新坐标。 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 销毁 | |
| 平移 | 执行容器中所有控件坐标的相对平移 |
| 标识 | |
| Id | 设置容器中所有控件 ID |
| 状态 | |
| Enable | 启用容器中所有控件 |
| Disable | 禁用容器中所有控件 |
| Show | 显示容器中所有控件 |
| Hide | 隐藏容器中所有控件 |
| 鼠标操作 | |
| MouseFocusKill | 清除鼠标焦点 |
| 文件操作 | |
| Save | 保存容器信息至文件 |
| Load | 从文件加载容器信息 |
| 内部事件处理器 | |
| OnResize | "调整大小"事件处理器 |
| OnActivate | "激活"事件处理器 |
| OnDeactivate | "解除激活"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Create, Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

© 2000-2025, MetaQuotes Ltd.

# Destroy

销毁所有控件容器

```
virtual bool  Destroy()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMouseEvent

鼠标事件处理器。

```
virtual bool  OnMouseEvent(
   const int  x,          // x 坐标
   const int  y,          // y 坐标
   const int  flags       // 标志
   )
```

参数

*x*

[输入] 鼠标光标相对于图表左上角的 X 坐标。

*y*

[输入] 鼠标光标相对于图表左上角的 Y 坐标。

*flags*

[输入] 鼠标按钮状态标志。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ControlsTotal

获取容器中控件的数量。

```
int  ControlsTotal()  const
```

返回值

容器中控件的数量。

© 2000-2025, MetaQuotes Ltd.

# Control

获取容器内索引处的控件。

```
CWnd*  Control(
   const int  ind      // 索引
   )  const
```

参数

*ind*

[输入] 所需控件的索引。

返回值

控件指针，否则，若未发现控件 NULL。

© 2000-2025, MetaQuotes Ltd.

# ControlFind

获取容器内标识符对应的控件。

```
virtual CWnd*  ControlFind(
   const long  id      // id
   )
```

参数

*id*

[输入] 控件 ID。

返回值

控件指针，否则，若未发现控件 NULL。

© 2000-2025, MetaQuotes Ltd.

# Add

添加控件至容器。

```
bool  Add(
   CWnd&  control      // 引用
   )
```

参数

*control*

[输入] 添加的控件，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Delete

从容器中删除控件。

```
bool  Delete(
   CWnd&  control      // 引用
   )
```

参数

*control*

[输入] 删除的控件，通过引用传递。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Move

设置容器中所有控件新坐标。

```
virtual bool  Move(
   const int  x,      // x 坐标
   const int  y       // y 坐标
   )
```

参数

*x*

[输入] 左上角的新 X 坐标。

*y*

[输入] 左上角的新1 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Shift

执行容器中所有控件坐标的相对平移。

```
virtual bool  Shift(
   const int  dx,      // 增量 x
   const int  dy       // 增量 y
   )
```

参数

*dx*

[输入] 增量 X。

*dy*

[输入] 增量 Y。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Id

设置容器中所有控件 ID。

```
virtual long  Id(
   const long  id      // 标识符
   )
```

参数

*id*

[输入] 基准组标识符。

返回值

容器内控件使用的标识符数量。

© 2000-2025, MetaQuotes Ltd.

# Enable

启用容器中所有控件。

```
virtual bool  Enable()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Disable

禁用容器中所有控件。

```
virtual bool  Disable()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Show

显示容器中所有控件。

```
virtual bool  Show()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Hide

隐藏容器中所有控件。

```
virtual bool  Hide()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# MouseFocusKill

清除保存的鼠标按钮状态并解除容器内控件激活。

```
bool  MouseFocusKill(
   const long  id=CONTROLS_INVALID_ID      // id
   )
```

参数

*id=CONTROLS_INVALID_ID*

[输入] 接收鼠标焦点的控件标识符。

返回值

控件解除激活结果。

© 2000-2025, MetaQuotes Ltd.

## Save

保存容器信息至文件。

```
virtual bool  Save(
   const int  file_handle      // 句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄 (必须以写模式打开)。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Load

从文件加载容器信息

```
virtual bool  Load(
   const int  file_handle      // 句柄
   )
```

参数

*file_handle*

[输入] 二进制文件句柄 (必须以读模式打开)。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnActivate

控件的"激活"事件处理器。

```
virtual bool  OnActivate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnDeactivate

控件的"解除激活"事件处理器。

```
virtual bool  OnDeactivate()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# CLabel

CLabel是一个简单控件，基于"文本标签"图表对象。

## 描述

CBmpButton的目的在于创建简单的文本标签。

## 声明

```
class CLabel : public CWndObj
```

## 标称库文件

```
#include <Controls\Label.mqh>
```

## 继承体系

CObject
  CWnd
    CWndObj
      CLabel

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| 属性改变事件处理器 | |
| OnSetText | "设置文本"事件处理器 |
| OnSetColor | "设置颜色"事件处理器 |
| OnSetFont | "设置字体"事件处理器 |
| OnSetFontSize | "设置字号"事件处理器 |
| 内部事件处理器 | |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建文本标签面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsLabel.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CLabel"
#include <Controls\Dialog.mqh>
#include <Controls\Label.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CLabel            m_label;                        // CLabel object
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
protected:
   //--- create dependent controls
   bool              CreateLabel(void);
   //--- handlers of the dependent controls events
   void              OnClickLabel(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)

EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateLabel())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CLabel"                                              |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateLabel(void)
  {
//--- coordinates
   int x1=INDENT_RIGHT;
   int y1=INDENT_TOP+CONTROLS_GAP_Y;
   int x2=x1+100;
   int y2=y1+20;
//--- create
   if(!m_label.Create(m_chart_id,m_name+"Label",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_label.Text("Label"))
      return(false);
   if(!Add(m_label))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CLabel控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetText

控件的"设置文本"(改变 OBJPROP_TEXT 属性)事件处理器。

```
virtual bool  OnSetText()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColor

控件的"设置颜色"(改变 OBJPROP_COLOR 属性)事件处理器。

```
virtual bool  OnSetColor()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFont

控件的"设置字体"(改变 OBJPROP_FONT 属性)事件处理器。

```
virtual bool  OnSetFont()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFontSize

控件的"设置字号"(改变 OBJPROP_FONTSIZE 属性)事件处理器。

```
virtual bool  OnSetFontSize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CBmpButton

CBmpButton 是一个简单控件，基于 "位图标签" 图表对象。

## 描述

CBmpButton 的目的在于创建图形图像按钮。

## 声明

```
class CBmpButton : public CWndObj
```

## 标称库文件

```
#include <Controls\BmpButton.mqh>
```

## 继承体系

COBject
CWnd
CWndObj
CBmpButton

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| 属性 | |
| Border | 获取/设置控件的"边框"属性 |
| BmpNames | 设置控件位图的文件名 |
| BmpOffName | 获取/设置关闭状态的位图文件名 |
| BmpOnName | 获取/设置打开状态的位图文件名 |
| BmpPassiveName | 获取/设置被动状态的位图文件名 |
| BmpActiveName | 获取/设置激活状态的位图文件名 |
| 状态 | |
| Pressed | 获取/设置控件的状态 |
| Locking | 获取/设置控件的"锁定"属性 |
| 内部事件处理器 | |
| OnSetZOrder | "设置图层顺序"事件处理器 |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnChange | "修改"事件处理器 |
| OnActivate | "激活"事件处理器 |
| OnDeactivate | "解除激活"事件处理器 |
| OnMouseDown | "鼠标按下"事件处理器 |
| OnMouseUp | "鼠标抬起"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建位图标签面板的示例：

```
//+------------------------------------------------------------------+
//|                                            ControlsBmpButton.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CBmpButton"
#include <Controls\Dialog.mqh>
#include <Controls\BmpButton.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CBmpButton        m_bmpbutton1;                  // CBmpButton object
   CBmpButton        m_bmpbutton2;                  // CBmpButton object
```

© 2000-2025, MetaQuotes Ltd.

```
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateBmpButton1(void);
   bool              CreateBmpButton2(void);
   //--- handlers of the dependent controls events
   void              OnClickBmpButton1(void);
   void              OnClickBmpButton2(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_bmpbutton1,OnClickBmpButton1)
ON_EVENT(ON_CLICK,m_bmpbutton2,OnClickBmpButton2)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateBmpButton1())
      return(false);
   if(!CreateBmpButton2())
      return(false);
//--- succeed
   return(true);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Create the "BmpButton1" button                                   |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateBmpButton1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_bmpbutton1.Create(m_chart_id,m_name+"BmpButton1",m_subwin,x1,y1,x2,y2))
      return(false);
//--- sets the name of bmp files of the control CBmpButton
   m_bmpbutton1.BmpNames("\\Images\\euro.bmp","\\Images\\dollar.bmp");
   if(!Add(m_bmpbutton1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "BmpButton2" fixed button                             |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateBmpButton2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_bmpbutton2.Create(m_chart_id,m_name+"BmpButton2",m_subwin,x1,y1,x2,y2))
      return(false);
//--- sets the name of bmp files of the control CBmpButton
   m_bmpbutton2.BmpNames("\\Images\\euro.bmp","\\Images\\dollar.bmp");
   if(!Add(m_bmpbutton2))
      return(false);
   m_bmpbutton2.Locking(true);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickBmpButton1(void)
  {
   Comment(__FUNCTION__);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickBmpButton2(void)
  {
   if(m_bmpbutton2.Pressed())
      Comment(__FUNCTION__+" State of the control is: On");
   else
      Comment(__FUNCTION__+" State of the control is: Off");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CBmpButton 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Border (Get方法)

获取控件的"边框"(边框宽度)属性。

```
int  Border()  const
```

返回值

该"边框"属性。

## Border (Set方法)

设置控件的"边框"(边框宽度)属性。

```
bool  Border(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"边框"属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# BmpNames

设置控件位图的文件名

```
bool  BmpNames(
   const string  off="",      // 文件名
   const string  on=""        // 文件名
   )
```

参数

*off=""*

[输入]关闭状态的位图文件名。

*on=""*

[输入]打开状态的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BmpOffName (Get方法)

获取关闭状态的位图文件名。

```
string  BmpOffName()  const
```

返回值

关闭状态的位图文件名。

## BmpOffName (Set方法)

设置关闭状态的位图文件名。

```
bool  BmpOffName(
   const string  name      // 文件名
   )
```

参数

*name*

[输入]关闭状态的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BmpOnName (Get方法)

获取打开状态的位图文件名。

```
string  BmpOnName()  const
```

返回值

打开状态的位图文件名。

## BmpOnName (Set方法)

设置打开状态的位图文件名。

```
bool  BmpOnName(
   const string  name      // 文件名
   )
```

参数

*name*

[输入]打开状态的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BmpPassiveName (Get方法)

获取控件被动状态的位图文件名。

```
string  BmpPassiveName()  const
```

返回值

控件被动状态的位图文件名。

## BmpPassiveName (Set方法)

设置被动状态的位图文件名。

```
bool  BmpPassiveName(
   const string  name      // 文件名
   )
```

参数

*name*

[输入]控件被动状态的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BmpActiveName (Get方法)

获取激活状态的位图文件名

```
string  BmpActiveName()  const
```

返回值

激活状态的位图文件名。

注释

当鼠标光标悬浮其上时控件激活。

## BmpActiveName (Set方法)

设置激活状态的位图文件名。

```
bool  BmpActiveName(
   const string  name      // 文件名
   )
```

参数

*name*

[输入]激活状态的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Pressed (Get方法)

获取控件的("按下"属性)状态。

```
bool  Pressed()  const
```

返回值

控件状态。

## Pressed (Set方法)

设置控件的("按下"属性)状态。

```
bool  Pressed(
   const bool  pressed      // 新状态
   )
```

参数

*pressed*

[输入] 新控件状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Locking (Get方法)

获取控件的"锁定"属性。

```
bool  Locking()  const
```

返回值

"锁定"属性的值。

## Locking (Set方法)

设置控件"锁定"属性的值。

```
void  Locking(
   const bool  locking      // 新值
   )
```

参数

*locking*

[输入]"锁定"属性的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnSetZOrder

控件的"设置图层顺序"(改变 OBJPROP_ZORDER 属性)事件处理器。

```
virtual bool  OnSetZOrder()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnActivate

控件的"激活"事件处理器。

```
virtual bool  OnActivate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnDeactivate

控件的"解除激活"事件处理器。

```
virtual bool  OnDeactivate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMouseDown

控件的"鼠标按下"事件处理器。

```
virtual bool  OnMouseDown()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上按鼠标左键时"鼠标按下"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnMouseUp

控件的"鼠标抬起"事件处理器。

```
virtual bool  OnMouseUp()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上释放鼠标左键时"鼠标抬起"事件触发。

© 2000-2025, MetaQuotes Ltd.

# CButton

CButton 是一个简单控件，基于"按钮"图表对象。

## 描述

CButton 的目的在于创建简单按钮。

## 声明

```
class CButton : public CWndObj
```

## 标称库文件

```
#include <Controls\Button.mqh>
```

## 继承体系

CObject
CWnd
CWndObj
CButton

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 状态 | |
| Pressed | 获取/设置"按下"属性 |
| Locking | 获取/设置"锁定"属性 |
| 属性改变事件处理器 | |
| OnSetText | "设置文本"事件处理器 |
| OnSetColor | "设置颜色"事件处理器 |
| OnSetColorBackground | "设置背景颜色"事件处理器 |
| OnSetColorBorder | "设置边框颜色"事件处理器 |
| OnSetFont | "设置字体"事件处理器 |
| OnSetFontSize | "设置字号"事件处理器 |
| 内部事件处理器 | |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnResize | "调整大小"事件处理器 |
| OnMouseDown | "鼠标按下"事件处理器 |
| OnOnMouseUp | "鼠标抬起"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建按键面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsButton.mq5 |
```

© 2000-2025, MetaQuotes Ltd.

```
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CButton"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CButton           m_button1;                     // the button object
   CButton           m_button2;                     // the button object
   CButton           m_button3;                     // the fixed button object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
```

```
   //--- create dependent controls
   bool              CreateButton1(void);
   bool              CreateButton2(void);
   bool              CreateButton3(void);
   //--- handlers of the dependent controls events
   void              OnClickButton1(void);
   void              OnClickButton2(void);
   void              OnClickButton3(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_button1,OnClickButton1)
ON_EVENT(ON_CLICK,m_button2,OnClickButton2)
ON_EVENT(ON_CLICK,m_button3,OnClickButton3)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateButton1())
      return(false);
   if(!CreateButton2())
      return(false);
   if(!CreateButton3())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button1" button                                      |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
bool CControlsDialog::CreateButton1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_button1.Create(m_chart_id,m_name+"Button1",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_button1.Text("Button1"))
      return(false);
   if(!Add(m_button1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button2" button                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButton2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_button2.Create(m_chart_id,m_name+"Button2",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_button2.Text("Button2"))
      return(false);
   if(!Add(m_button2))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button3" fixed button                                |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButton3(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_button3.Create(m_chart_id,m_name+"Button3",m_subwin,x1,y1,x2,y2))
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
   if(!m_button3.Text("Locked"))
      return(false);
   if(!Add(m_button3))
      return(false);
   m_button3.Locking(true);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton1(void)
  {
   Comment(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton2(void)
  {
   Comment(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton3(void)
  {
   if(m_button3.Pressed())
      Comment(__FUNCTION__+" State of the control: On");
   else
      Comment(__FUNCTION__+" State of the control: Off");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CButton 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Pressed (Get方法)

获取控件的("按下"属性)状态。

```
bool  Pressed()  const
```

返回值

控件状态。

## Pressed (Set方法)

设置控件的("按下"属性)状态。

```
bool  Pressed(
   const bool  pressed      // 新状态
   )
```

参数

*pressed*

[输入] 新控件状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Locking (Get方法)

获取控件的"锁定"属性。

```
bool  Locking()  const
```

返回值

"锁定"属性的值。

## Locking (Set方法)

设置控件"锁定"属性的值。

```
void  Locking(
   const bool  locking      // 新值
   )
```

参数

*locking*

[输入]"锁定"属性的新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnSetText

控件的"设置文本"(改变 OBJPROP_TEXT 属性)事件处理器。

```
virtual bool  OnSetText()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColor

控件的"设置颜色"(改变 OBJPROP_COLOR 属性)事件处理器。

```
virtual bool  OnSetColor()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBackground

控件的"设置背景颜色"(改变 OBJPROP_BGCOLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBorder

控件的"设置边框颜色"(改变 OBJPROP_BORDER_COLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFont

控件的"设置字体"(改变 OBJPROP_FONT 属性)事件处理器。

```
virtual bool  OnSetFont()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFontSize

控件的"设置字号"(改变 OBJPROP_FONTSIZE 属性)事件处理器。

```
virtual bool  OnSetFontSize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMouseDown

控件的"鼠标按下"事件处理器。

```
virtual bool  OnMouseDown()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上按鼠标左键时"鼠标按下"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnMouseUp

控件的"鼠标抬起"事件处理器。

```
virtual bool  OnMouseUp()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当在控件之上释放鼠标左键时"鼠标抬起"事件触发。

© 2000-2025, MetaQuotes Ltd.

# CEdit

CEdit是一个简单控件，基于"编辑"图表对象。

## 描述

CEdit类的目的在于创建用户可以输入文本的控件。

## 声明

```
class CEdit : public CWndObj
```

## 标称库文件

```
#include <Controls\Edit.mqh>
```

## 继承体系

CObject
  CWnd
    CWndObj
      CEdit

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 属性 | |
| ReadOnly | 获取/设置"只读"属性 |
| TextAlign | 获取/设置"文本对齐"属性 |
| 图表对象事件处理器 | |
| OnObjectEndEdit | 响应 CHARTEVENT_OBJECT_ENDEDIT 事件处理器（虚函数） |
| 属性改变事件处理器 | |
| OnSetText | "设置文本"事件处理器 |
| OnSetColor | "设置颜色"事件处理器 |
| OnSetColorBackground | "设置背景颜色"事件处理器 |
| OnSetColorBorder | "设置边框颜色"事件处理器 |
| OnSetFont | "设置字体"事件处理器 |
| OnSetFontSize | "设置字号"事件处理器 |
| OnSetZOrder | "设置图层顺序"事件处理器 |
| 内部事件处理器 | |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnResize | "调整大小"事件处理器 |
| OnChange | "修改"事件处理器 |
| OnClick | "点击"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建编辑控制面板的示例：

```
//+------------------------------------------------------------------+
//|                                                 ControlsEdit.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CEdit"
#include <Controls\Dialog.mqh>
#include <Controls\Edit.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CEdit             m_edit;                         // CEdit object

public:
```

© 2000-2025, MetaQuotes Ltd.

```
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler

protected:
   //--- create dependent controls
   bool              CreateEdit(void);
  };
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateEdit())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the display field                                         |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateEdit(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=ClientAreaWidth()-INDENT_RIGHT;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_edit.Create(m_chart_id,m_name+"Edit",m_subwin,x1,y1,x2,y2))
      return(false);
//--- allow editing the content
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!m_edit.ReadOnly(false))
      return(false);
   if(!Add(m_edit))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CEdit 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## ReadOnly (Get方法)

获取控件的"只读"属性。

```
bool  ReadOnly()
```

返回值

控件的"只读"属性值。

## ReadOnly (Set方法)

设置控件"只读"属性的值。

```
bool  ReadOnly(
   const bool  flag      // 新值
   )
```

参数

*flag*

[输入]"只读"属性的新值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## TextAlign (Get方法)

获取控件的"文本对齐"属性值 (文本对齐模式)。

```
ENUM_ALIGN_MODE  TextAlign() const
```

返回值

控件的"文本对齐"属性值。

## TextAlign (Set方法)

设置控件的"文本对齐"属性 (文本对齐模式) 的新值。

```
bool  TextAlign(
   ENUM_ALIGN_MODE  align      // 新值
   )
```

参数

*align*

[输入]"文本对齐"属性的新值。

返回值

true 如果成功, false 如果属性未改变。

© 2000-2025, MetaQuotes Ltd.

# OnObjectEndEdit

响应 CHARTEVENT_OBJECT_ENDEDIT 事件处理器。

```
virtual bool  OnObjectEndEdit()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetText

控件的"设置文本"(改变 OBJPROP_TEXT 属性)事件处理器。

```
virtual bool  OnSetText()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColor

控件的"设置颜色"(改变 OBJPROP_COLOR 属性)事件处理器。

```
virtual bool  OnSetColor()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBackground

控件的"设置背景颜色"(改变 OBJPROP_BGCOLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBorder

控件的"设置边框颜色"(改变 OBJPROP_BORDER_COLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFont

控件的"设置字体"(改变 OBJPROP_FONT 属性)事件处理器。

```
virtual bool  OnSetFont()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetFontSize

控件的"设置字号"(改变 OBJPROP_FONTSIZE 属性)事件处理器。

```
virtual bool  OnSetFontSize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetZOrder

控件的"设置图层顺序"(改变 OBJPROP_ZORDER 属性)事件处理器。

```
virtual bool  OnSetZOrder()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClick

控件的"点击"(鼠标左键点击)事件处理器。

```
virtual bool  OnClick()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CPanel

CPanel是一个简单控件，基于"长方形标签"图表对象。

## 描述

CPanel类的意图是将MQL5程序内部相似功能的控件结合到一组。

## 声明

```
class CPanel : public CWndObj
```

## 标称库文件

```
#include <Controls\Panel.mqh>
```

## 继承体系

CObject
CWnd
CWndObj
CPanel

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 图表对象属性 | |
| BorderType | 获取图表对象的"边框类型"属性。 |
| 图表对象事件处理器 | |
| OnSetText | "设置文本"事件处理器 |
| OnSetColorBackground | "设置背景颜色"事件处理器 |
| OnSetColorBorder | "设置边框颜色"事件处理器 |
| 内部事件处理器 | |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnResize | "调整大小"事件处理器 |
| OnChange | "修改"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建矩形标签面板的示例：

```
//+------------------------------------------------------------------+
//|                                                 ControlsPanel.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPanel"
```

© 2000-2025, MetaQuotes Ltd.

```
#include <Controls\Dialog.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const

protected:
   //--- create dependent controls
   bool              CreatePanel(void);
  };
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreatePanel())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CPanel"                                              |
//+------------------------------------------------------------------+
bool CControlsDialog::CreatePanel(void)
  {
//--- coordinates
   int x1=20;
   int y1=20;
   int x2=ExtDialog.Width()/3;
   int y2=ExtDialog.Height()/3;
//--- create
   if(!my_white_border.Create(0,ExtDialog.Name()+"MyWhiteBorder",m_subwin,x1,y1,x2,y2)
      return(false);
   if(!my_white_border.ColorBackground(CONTROLS_DIALOG_COLOR_BG))
      return(false);
   if(!my_white_border.ColorBorder(CONTROLS_DIALOG_COLOR_BORDER_LIGHT))
      return(false);
   if(!ExtDialog.Add(my_white_border))
      return(false);
   my_white_border.Alignment(WND_ALIGN_CLIENT,0,0,0,0);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//---
CPanel   my_white_border;        // object CPanel
bool     pause=true;             // true - pause
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
```

© 2000-2025, MetaQuotes Ltd.

```
   EventSetTimer(3);
   pause=true;
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
//+------------------------------------------------------------------+
//|  Timer                                                           |
//+------------------------------------------------------------------+
void OnTimer()
  {
   pause=!pause;
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CPanel控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BorderType (Get方法)

获取图表对象的"边框类型"属性。

```
ENUM_BORDER_TYPE  BorderType()
```

返回值

"边框类型"属性的值。

## BorderType (Set方法)

设置图表对象"边框类型"属性的新值。

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type      // 新值
   )
```

参数

*type*

[输入]"边框类型"属性的新值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetText

控件的"设置文本"(改变 OBJPROP_TEXT 属性)事件处理器。

```
virtual bool  OnSetText()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBackground

控件的"设置背景颜色"(改变 OBJPROP_BGCOLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnSetColorBorder

控件的"设置边框颜色"(改变 OBJPROP_BORDER_COLOR 属性)事件处理器。

```
virtual bool  OnSetColorBackground()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CPicture

CPicture 是一个简单控件，基于 "位图标签" 图表对象。

## 描述

CPicture 的目的在于创建简单的图形图像。

## 声明

```
class CPicture : public CWndObj
```

## 标称库文件

```
#include <Controls\Picture.mqh>
```

## 继承体系

COBject
CWnd
CWndObj
CPicture

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 图表对象属性 | |
| Border | 获取/设置图表对象的边框宽度 |
| BmpName | 获取/设置控件的位图文件名 |
| 内部事件 | |
| OnCreate | "创建"事件处理器 |
| OnShow | "显示"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnMove | "移动"事件处理器 |
| OnChange | "修改"事件处理器 |

方法继承自类CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

方法继承自类CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

创建位图标签面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsPicture.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPicture"
#include <Controls\Dialog.mqh>
#include <Controls\Picture.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
```

© 2000-2025, MetaQuotes Ltd.

```
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CPicture          m_picture;                      // CPicture object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreatePicture(void);
   //--- handlers of the dependent controls events
   void              OnClickPicture(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_picture,OnClickPicture)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreatePicture())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Picture"                                             |
//+------------------------------------------------------------------+
bool CControlsDialog::CreatePicture(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+32;
   int y2=y1+32;
//--- create
   if(!m_picture.Create(m_chart_id,m_name+"Picture",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set the name of bmp files to display the CPicture control
   m_picture.BmpName("\\Images\\euro.bmp");

   if(!Add(m_picture))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickPicture(void)
  {
   Comment(__FUNCTION__);
  }
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CPicture 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Border (Get方法)

获取控件的"边框"(边框宽度)属性。

```
int  Border()  const
```

返回值

该"边框"属性。

## Border (Set方法)

设置控件的"边框"(边框宽度)属性。

```
bool  Border(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"边框"属性值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## BmpName (Get方法)

获取控件的位图文件名。

```
string  BmpName()  const
```

返回值

控件位图的文件名。

## BmpName (Set方法)

设置控件的位图文件名。

```
bool  BmpName(
   const string  name      // 文件名
   )
```

参数

*name*

[输入]控件的位图文件名。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnCreate

控件的"创建"事件处理器。

```
virtual bool  OnCreate()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnMove

控件的"移动"事件处理器。

```
virtual bool  OnMove()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChange

控件的"改变"事件处理器。

```
virtual bool  OnChange()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CScroll

CCanvas 是创建滚动条的基准类。

## 描述

CScroll 是一个复合控件（含依赖控件），它包括创建滚动条的基础功能。基准类本身不能用作单独控件，它的两个继承者（CScrollV 和 CScrollH 类）可用于控件。

## 声明

```
class CScroll : public CWndContainer
```

## 标称库文件

```
#include <Controls\Scrolls.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CScroll

直接分支

CScrollH, CScrollV

## 类方法

| | |
|---|---|
| 创建 | |
| 创建 | 创建控件 |
| 图表对象事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| MinPos | 获取/设置最小位置 |
| MaxPos | 获取/设置最大位置 |
| CurrPos | 获取/设置当前位置 |
| 依赖控件创建 | |
| CreateBack | 创建背景按钮 |
| CreateInc | 创建滚动条的递增按钮 |
| CreateDec | 创建滚动条的递减按钮 |
| CreateThumb | 创建滚动条的滑块按钮（可被拖动） |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| 依赖控件事件处理器 | |
| OnClickInc | 事件的处理器，用于处理递增按钮事件 |
| OnClickDec | 事件的处理器，用于处理递减按钮事件 |
| 内部事件处理器 | |
| OnShow | "创建"事件处理器 |
| OnHide | "隐藏"事件处理器 |
| OnChangePos | "位置变化"事件处理器 |
| 对象拖动处理器 | |
| OnThumbDragStart | "滑块拖动开始"事件处理器 |
| OnThumbDragProcess | "滑块拖动过程"事件处理器 |
| OnThumbDragEnd | "滑块拖动结束"事件处理器 |
| 位置 | |
| CalcPos | 获取坐标处的滚动条位置 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CScroll控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## MinPos (Get方法)

获取CScroll控件的"最小位置"属性值。

```
int  MinPos()  const
```

返回值

新的"最小位置"属性值。

## MinPos (Set方法)

设置CScroll控件的"最小位置"属性值。

```
void  MinPos(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"最小位置"属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## MaxPos (Get方法)

获取 CScroll控件的 "最大位置" 属性值。

```
int  MaxPos()  const
```

返回值

新的 "最大位置" 属性值。

## MaxPos (Set方法)

设置 CScroll控件的 "最大位置" 属性值。

```
void  MaxPos(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的 "最大位置" 属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## CurrPos (Get方法)

获取CScroll控件的"当前位置"属性值。

```
int  CurrPos()  const
```

返回值

新的"当前位置"属性值。

## CurrPos (Set方法)

设置CScroll控件的"当前位置"属性值。

```
void  CurrPos(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"当前位置"属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CreateBack

创建 CScroll控件的背景按钮。

```
virtual bool  CreateBack()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateInc

创建 CScroll控件的递增按钮。

```
virtual bool  CreateInc()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateDec

创建 CScroll控件的递减按钮。

```
virtual bool  CreateDec()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateThumb

创建 CScroll控件的滑块按钮 (可被拖动)。

```
virtual bool  CreateThumb()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickInc

控件的"点击递增"(鼠标左键在递增按钮上点击)事件处理器。

```
virtual bool  OnClickInc()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickDec

控件的"点击递减"(鼠标左键在递减按钮上点击)事件处理器。

```
virtual bool  OnClickDec()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnShow

控件的"显示"事件处理器。

```
virtual bool  OnShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnHide

控件的"隐藏"事件处理器。

```
virtual bool  OnHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangePos

控件的"位置改变"(位置变化)事件处理器。

```
virtual bool  OnChangePos()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragStart

控件的"滑块拖动开始"(拖动开始)事件处理器。

```
virtual bool  OnThumbDragStart()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当开始拖动操作时"滑块拖动开始"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragProcess

控件的"滑块拖动过程"事件处理器。

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x 坐标
   const int  y       // y 坐标
   )
```

参数

*x*

[输入] 当前鼠标光标的 X 坐标。

*y*

[输入] 当前鼠标光标的 Y 坐标。

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件移动 (滑块按钮) 时"滑块拖动过程"触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragEnd

控件的"滑块拖动结束"(拖动完成)事件处理器。

```
virtual bool  OnThumbDragEnd()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件的拖动操作（滑块按钮）完成时"滑块拖动结束"触发。

© 2000-2025, MetaQuotes Ltd.

# CalcPos

获取坐标处的滚动条位置。

```
virtual int  CalcPos(
   const int      coord      // 坐标
   )
```

参数

*coord*

[输入] 滚动条坐标。

返回值

滚动条位置。

© 2000-2025, MetaQuotes Ltd.

# CScrollV

CScrollV 是 "垂直滚动条" 复合控件之中的类。

## 描述

CScrollV 的目的在于创建垂直滚动条。

## 声明

```
class CScrollV : public CScroll
```

## 标称库文件

```
#include <Controls\Scrolls.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CScroll
        CScrollV

下面提供的代码的结果：

## 类方法

© 2000-2025, MetaQuotes Ltd.

| 依赖控件 | |
|---|---|
| CreateInc | 创建滚动条递增按钮 |
| CreateDec | 创建滚动条递减按钮 |
| CreateThumb | 创建滚动条滑块按钮（可被拖动） |
| 内部事件处理器 | |
| OnResize | "调整大小"事件处理器 |
| OnChangePos | "位置变化"事件处理器 |
| 拖动事件处理器 | |
| OnThumbDragStart | "滑块拖动开始"事件处理器 |
| OnThumbDragProcess | "滑块拖动过程"事件处理器 |
| OnThumbDragEnd | "滑块拖动结束"事件处理器 |
| 位置 | |
| CalcPos | 获取坐标处的滚动条位置 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

方法继承自类CScroll

Create, OnEvent, MinPos, MinPos, MaxPos, MaxPos, CurrPos, CurrPos

创建垂直滚动条面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsScrollV.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollV"
```

```
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CScrollV          m_scroll_v;                     // CScrollV object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateScrollV(void);
   //--- handlers of the dependent controls events
   void              OnScrollInc(void);
   void              OnScrollDec(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateScrollV())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the CScrollsV object                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateScrollV(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=x1+18;
   int y2=y1+LIST_HEIGHT;
//--- create
   if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollV",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set up the scrollbar
   m_scroll_v.MinPos(0);
//--- set up the scrollbar
   m_scroll_v.MaxPos(10);
   if(!Add(m_scroll_v))
      return(false);
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
```

© 2000-2025, MetaQuotes Ltd.

```
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollInc(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollDec(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
```

© 2000-2025, MetaQuotes Ltd.

```
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
 ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

© 2000-2025, MetaQuotes Ltd.

# CreateInc

创建控件的递增按钮。

```
virtual bool  CreateInc()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateDec

创建控件的递减按钮。

```
virtual bool  CreateDec()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateThumb

创建控件的滑块按钮 (可被拖动)。

```
virtual bool  CreateThumb()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangePos

控件的"位置改变"(位置变化)事件处理器。

```
virtual bool  OnChangePos()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragStart

控件的"滑块拖动开始"(拖动开始)事件处理器。

```
virtual bool  OnThumbDragStart()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当开始拖动操作时"滑块拖动开始"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragProcess

控件的"滑块拖动过程"事件处理器。

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x 坐标
   const int  y       // y 坐标
   )
```

参数

*x*

[输入] 当前鼠标光标的 X 坐标。

*y*

[输入] 当前鼠标光标的 Y 坐标。

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件移动 (滑块按钮) 时"滑块拖动过程"触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragEnd

控件的"滑块拖动结束"(拖动完成)事件处理器。

```
virtual bool  OnThumbDragEnd()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件的拖动操作（滑块按钮）完成时"滑块拖动结束"触发。

© 2000-2025, MetaQuotes Ltd.

# CalcPos

获取坐标处的滚动条位置。

```
virtual int  CalcPos(
   const int      coord      // 坐标
   )
```

参数

*coord*

[输入] 滚动条坐标。

返回值

滚动条位置。

© 2000-2025, MetaQuotes Ltd.

# CScrollH

CScrollH 是 "水平滚动条" 复合控件之中的类。

## 描述

CScrollH 的目的在于创建水平滚动条。

## 声明

```
class CScrollH : public CScroll
```

## 标称库文件

```
#include <Controls\Scrolls.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CScroll
        CScrollH

下面提供的代码的结果：

## 类方法

© 2000-2025, MetaQuotes Ltd.

| 依赖控件 | |
|---|---|
| CreateInc | 创建滚动条递增按钮 |
| CreateDec | 创建滚动条递减按钮 |
| CreateThumb | 创建滚动条滑块按钮（可被拖动） |
| 内部事件处理器 | |
| OnResize | "调整大小"事件处理器 |
| OnChangePos | "位置变化"事件处理器 |
| 拖动事件处理器 | |
| OnThumbDragStart | "滑块拖动开始"事件处理器 |
| OnThumbDragProcess | "滑块拖动过程"事件处理器 |
| OnThumbDragEnd | "滑块拖动结束"事件处理器 |
| 位置 | |
| CalcPos | 获取坐标处的滚动条位置 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

方法继承自类CScroll

Create, OnEvent, MinPos, MinPos, MaxPos, MaxPos, CurrPos, CurrPos

创建水平滚动条面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsScrollH.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollH"
```

```
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CScrollH          m_scroll_v;                     // CScrollH object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateScrollsH(void);
   //--- handlers of the dependent controls events
   void              OnScrollInc(void);
   void              OnScrollDec(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateScrollsH())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the CScrollsH object                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateScrollsH(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=x1+3*BUTTON_WIDTH;
   int y2=y1+18;
//--- create
   if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollsH",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set up the scrollbar
   m_scroll_v.MinPos(0);
//--- set up the scrollbar
   m_scroll_v.MaxPos(10);
   if(!Add(m_scroll_v))
      return(false);
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
```

© 2000-2025, MetaQuotes Ltd.

```
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollInc(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollDec(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
```

© 2000-2025, MetaQuotes Ltd.

```
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
 ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

© 2000-2025, MetaQuotes Ltd.

# CreateInc

创建控件的递增按钮。

```
virtual bool  CreateInc()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateDec

创建控件的递减按钮。

```
virtual bool  CreateDec()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateThumb

创建控件的滑块按钮 (可被拖动)。

```
virtual bool  CreateThumb()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangePos

控件的"位置改变"(位置变化)事件处理器。

```
virtual bool  OnChangePos()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragStart

控件的"滑块拖动开始"(拖动开始)事件处理器。

```
virtual bool  OnThumbDragStart()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当开始拖动操作时"滑块拖动开始"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragProcess

控件的"滑块拖动过程"事件处理器。

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x 坐标
   const int  y       // y 坐标
   )
```

参数

*x*

[输入] 当前鼠标光标的 X 坐标。

*y*

[输入] 当前鼠标光标的 Y 坐标。

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件移动 (滑块按钮) 时"滑块拖动过程"触发。

© 2000-2025, MetaQuotes Ltd.

# OnThumbDragEnd

控件的"滑块拖动结束"(拖动完成)事件处理器。

```
virtual bool  OnThumbDragEnd()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当滚动条控件的拖动操作 (滑块按钮) 完成时"滑块拖动结束"触发。

© 2000-2025, MetaQuotes Ltd.

# CalcPos

获取坐标处的滚动条位置。

```
virtual int  CalcPos(
   const int      coord      // 坐标
   )
```

参数

*coord*

[输入] 滚动条坐标。

返回值

滚动条位置。

© 2000-2025, MetaQuotes Ltd.

# CWndClient

CWndClient是"客户区域"复合控件之中的类(含依赖控件)。它是创建滚动条的基准类。

## 描述

CWndClient实现了在客户区域创建滚动条的功能。

## 声明

```
class CWndClient : public CWndContainer
```

## 标称库文件

```
#include <Controls\WndClient.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CWndClient

直接分支

CCheckGroup, CListView, CRadioGroup

## 类方法

| | |
|---|---|
| 创建 | |
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| ColorBackground | 设置背景颜色 |
| ColorBorder | 设置边框颜色 |
| BorderType | 设置边框类型 |
| 设置 | |
| VScrolled | 获取/设置垂直滚动条的使用指示标志 |
| HScrolled | 获取/设置水平滚动条的使用指示标志 |
| 依赖控件 | |
| CreateBack | 创建滚动条的背景 |
| CreateScrollV | 创建垂直滚动条 |

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| CreateScrollH | 创建水平滚动条 |
| 内部事件处理器 | |
| OnResize | "调整大小"事件处理器 |
| 依赖控件事件处理器 | |
| OnVScrollShow | VScroll依赖控件的"显示"事件处理器（虚函数） |
| OnVScrollHide | VScroll依赖控件的"隐藏"事件处理器（虚函数） |
| OnHScrollShow | HScroll依赖控件的"显示"事件处理器（虚函数） |
| OnHScrollHide | HScroll依赖控件的"隐藏"事件处理器（虚函数） |
| OnScrollLineDown | VScroll依赖控件的"滚动条向下"事件处理器（虚函数） |
| OnScrollLineUp | VScroll依赖控件的"滚动条向上"事件处理器（虚函数） |
| OnScrollLineLeft | HScroll依赖控件的"左滚动线"事件处理器（虚函数） |
| OnScrollLineRight | HScroll依赖控件的"右滚动线"事件处理器（虚函数） |
| 调整大小 | |
| Rebound | 设置 CRect 类的新坐标 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide, Save, Load

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CWndClient 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ColorBackground

设置控件的背景颜色。

```
bool  ColorBackground(
   const color  value      // 新颜色
   )
```

参数

*value*

[输入] 控件的新背景颜色。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ColorBorder

设置控件的边框颜色。

```
bool  ColorBorder(
   const color  value      // 颜色
   )
```

参数

*value*

[输入] 控件的新边框颜色。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# BorderType

设置控件的边框类型。

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type      // 边框类型
   )
```

参数

*type*

[输入] 控件的边框类型。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## VScrolled (Get方法)

获取垂直滚动条的使用指示标志。

```
bool  VScrolled()
```

返回值

true，如果垂直滚动条被使用，否则 false。

## VScrolled (Set方法)

设置垂直滚动条的使用指示标志。

```
bool  VScrolled(
   const bool  flag      // 标志
   )
```

参数

*flag*

[输入] 标志。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## HScrolled (Get方法)

获取水平滚动条的使用指示标志。

```
bool  HScrolled()
```

返回值

true, 如果水平滚动条被使用, 否则 false。

## HScrolled (Set方法)

设置水平滚动条的使用指示标志。

```
bool  HScrolled(
   const bool  flag      // 标志
   )
```

参数

*flag*

[输入] 标志。

返回值

true 如果成功, 否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateBack

创建控件的背景按钮。

```
virtual bool  CreateBack()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateScrollV

创建垂直滚动条。

```
virtual bool  CreateScrollV()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateScrollH

创建水平滚动条。

```
virtual bool  CreateScrollH()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollShow

控件的"垂直滚动显示"(垂直滚动条显示)事件处理器。

```
virtual bool  OnVScrollShow()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollHide

控件的"垂直滚动隐藏"(垂直滚动条隐藏)事件处理器。

```
virtual bool  OnVScrollHide()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnHScrollShow

控件的"水平滚动显示"(水平滚动条显示)事件处理器。

```
virtual bool  OnHScrollShow()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnHScrollHide

控件的"水平滚动隐藏"(水平滚动条隐藏)事件处理器。

```
virtual bool  OnHScrollHide()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineDown

控件的"滚动条向下"(垂直滚动条向下)事件处理器。

```
virtual bool  OnScrollLineDown()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineUp

控件的"滚动条向上"(垂直滚动条向上)事件处理器。

```
virtual bool  OnScrollLineUp()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineLeft

控件的"左侧滚动线"(左侧水平滚动线)事件处理器。

```
virtual bool  OnScrollLineLeft()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineRight

控件的"右侧滚动线"(右侧水平滚动线)事件处理器。

```
virtual bool  OnScrollLineRight()
```

返回值

true 如果事件已经被处理，否则 false。

注释

基准类中的方法不做任何事且永远返回 true。

© 2000-2025, MetaQuotes Ltd.

# ReBound

设置 CRect 类的新坐标。

```
void  ReBound(
   const & CRect  rect      // CRect 类
   )
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CListView

CListView 是 ListView 复合控件之中的类(含依赖控件)。

## 描述

CListView 类封装了列表控制功能。

## 声明

```
class CListView : public CWndClient
```

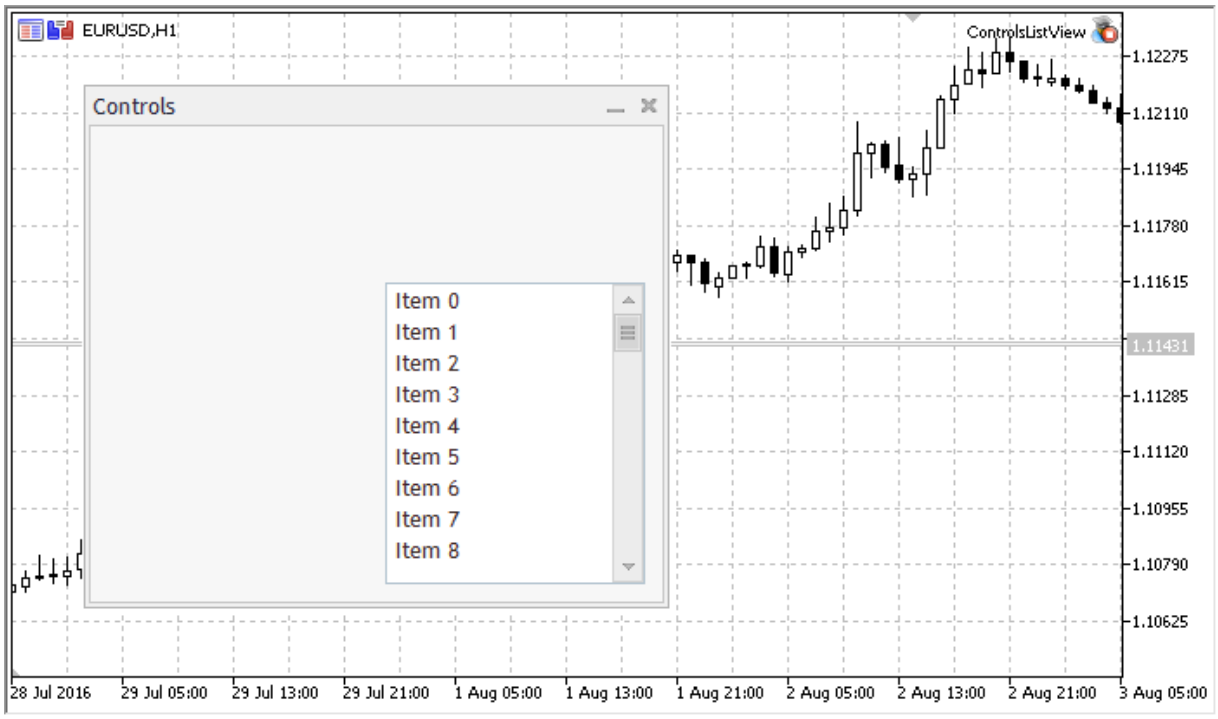
## 标称库文件

```
#include <Controls\ListView.mqh>
```

## 继承体系

COBject
CWnd
CWndContainer
CWndClient
CListView

下面提供的代码的结果：




## 类方法

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 设置 | |
| TotalView | 设置显示在控件上的项目数量 |
| 添加/删除 | |
| AddItem | 添加一个项目 |
| 数据 | |
| Select | 按照索引选择当前列表项 |
| SelectByText | 按照文本选择当前列表项 |
| SelectByValue | 按照值选择当前列表项 |
| 只读数据 | |
| 数值 | 获取当前列表项的值 |
| 依赖控件 | |
| CreateRow | 创建新的一行 ListView |
| 内部事件处理器 | |
| OnResize | "调整大小"事件处理器（虚函数） |
| 依赖控件事件处理器 | |
| OnVScrollShow | VScroll依赖控件的"显示"事件处理器（虚函数） |
| OnVScrollHide | VScroll依赖控件的"隐藏"事件处理器（虚函数） |
| OnScrollLineDown | VScroll依赖控件的"滚动条向下"事件处理器（虚函数） |
| OnScrollLineUp | VScroll依赖控件的"滚动条向上"事件处理器（虚函数） |
| OnItemClick | "点击项目"事件处理器（虚函数） |
| 重绘 | |
| 重绘 | 重绘控件 |
| RowState | 设置指定行的状态 |
| CheckView | 检查指定行的"可视性" |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide, Save, Load

方法继承自类CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

创建列表视图控制面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsListView.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                                https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CListView"
#include <Controls\Dialog.mqh>
#include <Controls\ListView.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
```

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CListView         m_list_view;                     // CListView object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateListView(void);
   //--- handlers of the dependent controls events
   void              OnChangeListView(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_list_view,OnChangeListView)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
```

© 2000-2025, MetaQuotes Ltd.

```
   if(!CreateListView())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "ListView" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateListView(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+GROUP_WIDTH+2*CONTROLS_GAP_X;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+2*CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+LIST_HEIGHT-CONTROLS_GAP_Y;
//--- create
   if(!m_list_view.Create(m_chart_id,m_name+"ListView",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_list_view))
      return(false);
//--- fill out with strings
   for(int i=0;i<16;i++)
      if(!m_list_view.AddItem("Item "+IntegerToString(i)))
         return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeListView(void)
  {
   Comment(__FUNCTION__+" \""+m_list_view.Select()+"\"");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
```

© 2000-2025, MetaQuotes Ltd.

```
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CListView 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# TotalView

设置显示在控件上的项目数量。

```
bool  TotalView(
   const int  value      // 项目显示
   )
```

参数

*value*

[输入]显示在控件上的项目数量。

返回值

true 如果成功，否则 false。

注释

显示项目数量仅能指定一次。

© 2000-2025, MetaQuotes Ltd.

# AddItem

添加新项目至控件。

```
bool  AddItem(
   const string  item,     // 文本
   const long    value     // 值
   )
```

参数

*item*

[输入] 文本。

*value*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Select

按照索引选择当前列表项。

```
bool  Select(
   const int  index     // 索引
   )
```

参数

*index*

[输入] 项目索引。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# SelectByText

按照文本选择当前列表项。

```
bool  SelectByText(
   const string  text      // 文本
   )
```

参数

*text*

[输入] 文本。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# SelectByValue

按照值选择当前列表项。

```
bool  SelectByValue(
   const long  value     // 值
   )
```

参数

*value*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Value

获取当前列表项的值。

```
long  Value()
```

返回值

当前列表项的值。

© 2000-2025, MetaQuotes Ltd.

# CreateRow

创建一行新的 CListView 控件。

```
bool  CreateRow(
   const int  index     // 索引
   )
```

参数

*index*

[输入] 项目索引。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnResize

控件的"调整大小"事件处理器。

```
virtual bool  OnResize()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollShow

控件的"垂直滚动显示"(垂直滚动条显示)事件处理器。

```
virtual bool  OnVScrollShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollHide

控件的"垂直滚动隐藏"(垂直滚动条隐藏)事件处理器。

```
virtual bool  OnVScrollHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineDown

控件的"滚动条向下"(垂直滚动条向下)事件处理器。

```
virtual bool  OnScrollLineDown()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineUp

控件的"滚动条向上"(垂直滚动条向上)事件处理器。

```
virtual bool  OnScrollLineUp()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnItemClick

控件的"点击项目"(鼠标点击一行)事件处理器。

```
virtual bool  OnItemClick()
   const int    index     // 行索引
   )
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Redraw

重绘控件

```
bool  Redraw()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# RowState

设置指定行的状态。

```
bool  RowState(
   const int   index      // 索引
   const bool  select     // 状态
   )
```

参数

*index*

[输入] 行索引。

*select*

[输入] 行状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CheckView

检查指定行的"可视性"。

```
bool  CheckView()
```

返回值

true，如果选择行可视，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CComboBox

CCom boBox 是 Com boBox 复合控件之中的类(含依赖控件)。

## 描述

Com boBox 由列表框构成，结合静态控件，目的用于选择。当用户选择下拉键头时，控件的列表框部分可以向下展开。

## 声明

```
class CComboBox : public CWndContainer
```

## 标称库文件

```
#include <Controls\ComboBox.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CComboBox

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 添加 | |
| AddItem | 添加一个项目 |
| 设置 | |
| ListViewItems | 设置显示在控件上的项目数量 |
| 数据 | |
| Select | 按照索引选择当前列表项 |
| SelectByText | 按照文本选择当前列表项 |
| SelectByValue | 按照值选择当前列表项 |
| 只读数据 | |
| 数值 | 获取当前列表项的值 |
| 依赖控件 | |
| CreateEdit | 创建依赖控件（编辑） |
| CreateButton | 创建依赖控件（按钮） |
| CreateList | 创建依赖控件（列表视野） |
| 依赖控件事件处理器 | |
| OnClickEdit | "点击编辑"事件处理器（虚函数） |
| OnClickButton | "点击按钮"事件处理器（虚函数） |
| OnChangeList | "改变列表"事件处理器（虚函数） |
| 显示/隐藏 | |
| ListShow | 显示项目列表 |
| ListHide | 隐藏项目列表 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Hide

创建组合框控制面板的示例：

```
//+------------------------------------------------------------------+
//|                                                ControlsComboBox.mq5 |
//|                        Copyright 2015, MetaQuotes Software Corp. |
//|                                                https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2015, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CComboBox"
#include <Controls\Dialog.mqh>
#include <Controls\ComboBox.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CComboBox         m_combo_box;;                    // CComboBox object

public:
```

© 2000-2025, MetaQuotes Ltd.

```
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateComboBox(void);
   //--- handlers of the dependent controls events
   void              OnChangeComboBox(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_combo_box,OnChangeComboBox)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateComboBox())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "ComboBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateComboBox(void)
  {
```

© 2000-2025, MetaQuotes Ltd.

```
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_combo_box.Create(m_chart_id,m_name+"ComboBox",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_combo_box))
      return(false);
//--- fill out with strings
   for(int i=0;i<16;i++)
      if(!m_combo_box.ItemAdd("Item "+IntegerToString(i)))
         return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeComboBox(void)
  {
   Comment(__FUNCTION__+" \""+m_combo_box.Select()+"\"");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
```

© 2000-2025, MetaQuotes Ltd.

```
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CCom boBox 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# AddItem

添加新项目至控件。

```
bool  AddItem(
   const string  item,     // 文本
   const long    value     // 值
   )
```

参数

*item*

[输入] 文本。

*value=0*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ListViewItems

设置 CComboBox 控件的列表项目数量。

```
void  ListViewItems(
   const int     value      // 列表项目数量
   )
```

参数

*value*

[输入] 列表项目数量。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Select

按照索引选择当前列表项。

```
bool  Select(
   const int  index     // 索引
   )
```

参数

*index*

[输入] 项目索引。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# SelectByText

按照文本选择当前列表项。

```
bool  SelectByText(
   const string  text      // 文本
   )
```

参数

*text*

[输入] 项目文本。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# SelectByValue

按照值选择当前列表项。

```
bool  SelectByValue(
   const long  value     // 值
   )
```

参数

*value*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Value

获取当前列表项的值。

```
long  Value()
```

返回值

当前列表项的值。

© 2000-2025, MetaQuotes Ltd.

# CreateEdit

创建依赖控件（编辑）。

```
virtual bool  CreateEdit()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateButton

创建依赖控件（按钮）。

```
virtual bool  CreateButton()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateList

创建依赖控件（列表视野）。

```
virtual bool  CreateList()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickEdit

控件的"点击编辑"(鼠标在编辑上点击)事件处理器。

```
virtual bool  OnClickEdit()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickButton

控件的"点击按钮"(鼠标在按钮上点击)事件处理器。

```
virtual bool  OnClickButton()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangeList

"列表改变"(列表有变化)事件处理器。

```
virtual bool  OnChangeList()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ListShow

显示项目列表。

```
virtual bool  ListShow()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ListHide

隐藏项目列表。

```
virtual bool  ListHide()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CCheckBox

CCheckBox 是 CheckBox 复合控件之中的类。

## 描述

CCheckBox 控件显示一个勾选框，以便用户选择一个真伪条件。

## 声明

```
class CCheckBox : public CWndContainer
```

## 标称库文件

```
#include <Controls\CheckBox.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CCheckBox

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| 文本 | 获取/设置控件关联的文本标签 |
| Color | 获取/设置控件关联的文本标签颜色 |
| 状态 | |
| Checked | 获取/设置控件是否被选择的指示值 |
| 数据 | |
| 数值 | 获取/设置控件关联的值 |
| 依赖控件 | |
| CreateButton | 创建依赖控件（按钮） |
| CreateLabel | 创建依赖控件（标签） |
| 依赖控件事件处理器 | |
| ClickButton | "点击按钮"事件处理器（虚函数） |
| ClickLabel | "点击标签"事件处理器（虚函数） |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

创建复选框控制面板的示例：

```
//+------------------------------------------------------------------+
//|                                            ControlsCheckBox.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
```

© 2000-2025, MetaQuotes Ltd.

```
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckBox"
#include <Controls\Dialog.mqh>
#include <Controls\CheckBox.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CCheckBox         m_check_box1;                    // CCheckBox object
   CCheckBox         m_check_box2;                    // CCheckBox object
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
protected:
   //--- create dependent controls
   bool              CreateCheckBox1(void);
   bool              CreateCheckBox2(void);
   //--- handlers of the dependent controls events
   void              OnChangeCheckBox1(void);
   void              OnChangeCheckBox2(void);
  };
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_box1,OnChangeCheckBox1)
ON_EVENT(ON_CHANGE,m_check_box2,OnChangeCheckBox2)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateCheckBox1())
      return(false);
   if(!CreateCheckBox2())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckBox1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_check_box1.Create(m_chart_id,m_name+"CheckBox1",m_subwin,x1,y1,x2,y2))
```

© 2000-2025, MetaQuotes Ltd.

```
      return(false);
   if(!m_check_box1.Text("CheckBox1"))
      return(false);
   if(!m_check_box1.Color(clrBlue))
      return(false);
   if(!Add(m_check_box1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckBox2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+GROUP_WIDTH+CONTROLS_GAP_X;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_check_box2.Create(m_chart_id,m_name+"CheckBox2",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_check_box2.Text("CheckBox2"))
      return(false);
   if(!m_check_box2.Color(clrBlue))
      return(false);
   if(!Add(m_check_box2))
      return(false);
   m_check_box2.Checked(true);
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckBox1(void)
  {
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box1.Checked()));
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckBox2(void)
```

© 2000-2025, MetaQuotes Ltd.

```
  {
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CCheckBox 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Text (Get 方法)

获取控件关联的标签文本。

```
string  Text()
```

返回值

标签文本。

## Text (Set 方法)

设置控件关联的标签文本。

```
bool  Text(
   const string  value      // 文本
   )
```

参数

*value*

[输入] 新的标签文本。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Color (Get方法)

获取控件关联的标签颜色。

```
color  Color()  const
```

返回值

标签颜色。

## Color (Set方法)

设置控件关联的标签颜色。

```
bool  Color(
   const color  value      // 颜色
   )
```

参数

*value*

[输入] 新标签颜色。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Checked (Get方法)

获取控件的状态。

```
bool  Checked()  const
```

返回值

控件的状态。

## Checked (Set方法)

设置控件的状态。

```
bool  Checked(
   const bool  flag      // 状态
   )
```

参数

*flag*

[输入] 新状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Value (Get方法)

获取控件关联的值。

```
int  Value()  const
```

返回值

控件关联的值。

## Value (Set方法)

设置控件关联的值。

```
void  Value(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CreateButton

创建依赖控件（按钮）。

```
virtual bool  CreateButton()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateLabel

创建依赖控件（标签）。

```
virtual bool  CreateLabel()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickButton

控件的"点击按钮"(鼠标在按钮上点击)事件处理器。

```
virtual bool  OnClickButton()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickLabel

控件的"点击标签"(鼠标在标签上点击)事件处理器。

```
virtual bool  OnClickLabel()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CCheckGroup

CCheckGroup 是 CheckGroup 复合控件之中的类(含依赖控件)。

## 描述

CCheckGroup 可供用户创建显示并编辑标志的控件。

## 声明

```
class CCheckGroup : public CWndClient
```

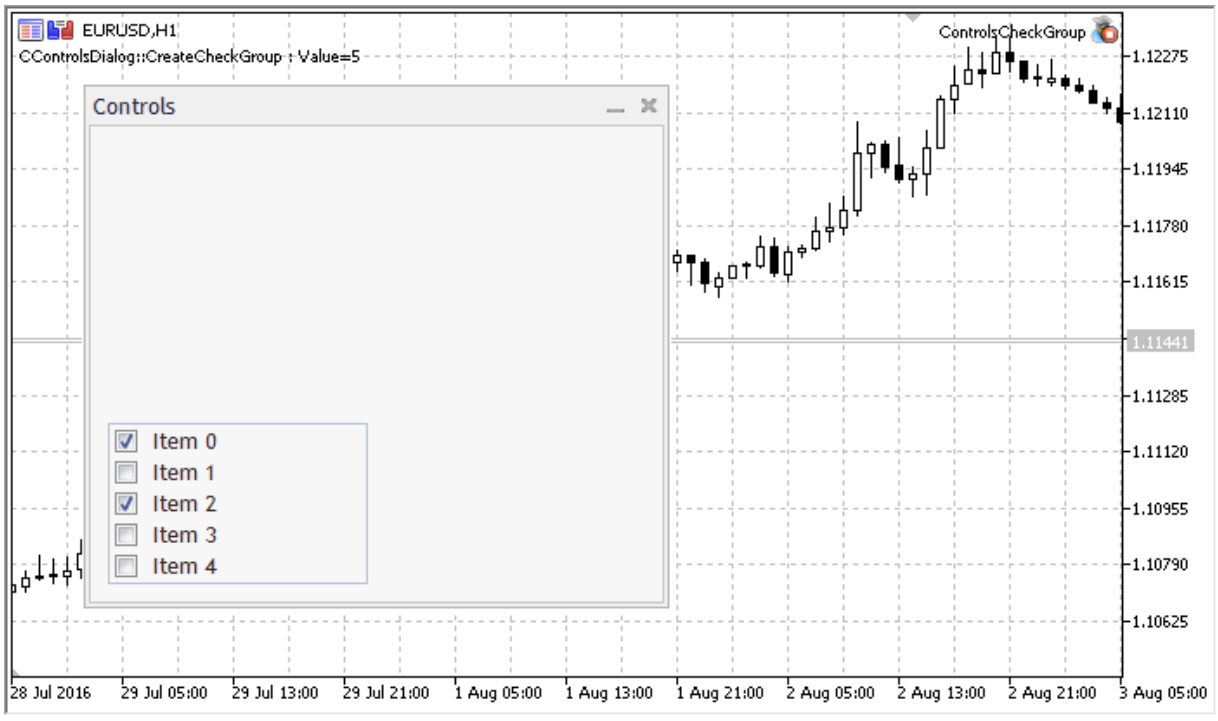
## 标称库文件

```
#include <Controls\CheckGroup.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CWndClient
        CCheckGroup

下面提供的代码的结果：




## 类方法

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的处理器 |
| 添加 | |
| AddItem | 添加新项目 |
| 只读数据 | |
| 数值 | 获取控件关联的值。 |
| 依赖控件 | |
| CreateButton | 创建新的 CCheckBox 控件。 |
| 依赖控件事件处理器 | |
| OnVScrollShow | VScroll依赖控件的"显示"事件处理器（虚函数） |
| OnVScrollHide | VScroll依赖控件的"隐藏"事件处理器（虚函数） |
| OnScrollLineDown | VScroll依赖控件的"滚动条向上"事件处理器（虚函数） |
| OnScrollLineUp | VScroll依赖控件的"滚动条向下"事件处理器（虚函数） |
| OnChangeItem | VScroll依赖控件的"项目改变"事件处理器（虚函数） |
| 重绘 | |
| 重绘 | 重绘控件组 |
| RowState | 设置指定项目的状态 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide

方法继承自类CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

创建复选框组控制面板的示例：

© 2000-2025, MetaQuotes Ltd.

```
//+------------------------------------------------------------------+
//|                                              ControlsCheckGroup.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckGroup"
#include <Controls\Dialog.mqh>
#include <Controls\CheckGroup.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CCheckGroup       m_check_group;                  // CCheckGroup object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
```

© 2000-2025, MetaQuotes Ltd.

```
   //--- create dependent controls
   bool              CreateCheckGroup(void);
   //--- handlers of the dependent controls events
   void              OnChangeCheckGroup(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_group,OnChangeCheckGroup)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateCheckGroup())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckGroup" element                                  |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckGroup(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
```

© 2000-2025, MetaQuotes Ltd.

```
   int y2=y1+CHECK_HEIGHT;
//--- create
   if(!m_check_group.Create(m_chart_id,m_name+"CheckGroup",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_check_group))
      return(false);
//--- fill out with strings
   for(int i=0;i<5;i++)
      if(!m_check_group.AddItem("Item "+IntegerToString(i),1<<i))
         return(false);
   m_check_group.Check(0,1<<0);
   m_check_group.Check(2,1<<2);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckGroup(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(ChartID(),"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CCheckGroup 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# AddItem

添加新项目至控件。

```
bool  AddItem(
   const string  item,     // 文本
   const long    value     // 值
   )
```

参数

*item*

[输入] 文本。

*value=0*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Value

获取控件关联的值。

```
long  Value()
```

返回值

控件关联的值。

注释

该值依赖所有 CCheckGroup 控件中的项目状态。

© 2000-2025, MetaQuotes Ltd.

# CreateButton

在指定索引处创建新的 CCheckBox 类实例。

```
bool  CreateButton(
   int  index      // 索引
   )
```

参数

*index*

[输入] 在 CCheckGroup 中的新项目索引值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollShow

控件的"垂直滚动显示"(垂直滚动条显示)事件处理器。

```
virtual bool  OnVScrollShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollHide

控件的"垂直滚动隐藏"(垂直滚动条隐藏)事件处理器。

```
virtual bool  OnVScrollHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineDown

控件的"滚动条向下"(垂直滚动条向下)事件处理器。

```
virtual bool  OnScrollLineDown()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineUp

控件的"滚动条向上"(垂直滚动条向上)事件处理器。

```
virtual bool  OnScrollLineUp()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangeItem

控件的"项目改变"(项目变化)事件处理器。

```
virtual bool  OnChangeItem(
   const int  index      // 索引
   )
```

参数

*index*

[输入] 改变项目的索引.

返回值

true 如果事件已经被处理, 否则 false。

© 2000-2025, MetaQuotes Ltd.

# Redraw

重绘控件

```
bool  Redraw()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# RowState

设置指定项目的状态。

```
bool  RowState(
   const int   index,      // 项目索引
   const bool  select      // 状态
   )
```

参数

*index*

[输入] 即将改变的项目索引。

*select*

[输入] 新状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CRadioButton

CRadioButton 是 RadioButton 复合控件之中的类。

## 描述

CRadioButton 自身无法使用，它是用来创建 CRadioGroup 项目。

## 声明

```
class CRadioButton : public CWndContainer
```

## 标称库文件

```
#include <Controls\RadioButton.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CRadioButton

## 类方法

| | |
|---|---|
| 创建 | |
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的处理器 |
| 属性 | |
| 文本 | 获取/设置控件关联的文本标签 |
| Color | 获取/设置控件关联的文本标签颜色 |
| 状态 | |
| 状态 | 获取/设置状态 |
| 依赖控件 | |
| CreateButton | 创建按钮 |
| CreateLabel | 创建标签 |
| 依赖控件事件处理器 | |
| OnClickButton | "点击按钮"事件处理器（虚函数） |
| OnClickLabel | "点击标签"事件处理器（虚函数） |

© 2000-2025, MetaQuotes Ltd.

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CRadioButton 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Text (Get 方法)

获取控件关联的标签文本。

```
string  Text()
```

返回值

标签文本。

## Text (Set 方法)

设置控件关联的标签文本。

```
bool  Text(
   const string  value      // 文本
   )
```

参数

*value*

[输入] 新的标签文本。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Color (Get 方法)

获取控件关联的标签颜色。

```
color  Color()  const
```

返回值

标签颜色。

## Color (Set 方法)

设置控件关联的标签颜色。

```
bool  Color(
   const color  value      // 颜色
   )
```

参数

*value*

[输入] 新标签颜色。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

## State (Get方法)

获取按钮状态。

```
bool  State()  const
```

返回值

按钮状态。

## State (Set方法)

设置按钮状态。

```
bool  State(
   const bool  flag      // 标志
   )
```

参数

*flag*

[输入] 新的按钮状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateButton

创建按钮。

```
virtual bool  CreateButton()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateLabel

创建标签。

```
virtual bool  CreateLabel()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickButton

控件的"点击按钮"(鼠标点击)事件处理器。

```
virtual bool  OnClickButton()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickLabel

控件的"点击标签"(鼠标在标签上点击)事件处理器。

```
virtual bool  OnClickLabel()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CRadioGroup

CRadioGroup 是 RadioGroup 复合控件之中的类(含依赖控件)。

## 描述

CRadioGroup 允许用户从一组可选项之中选择单一选项，需要与多个 CRadioButton 控件配对。

## 声明

```
class CRadioGroup : public CWndClient
```

## 标称库文件

```
#include <Controls\RadioGroup.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CWndClient
        CRadioGroup

下面提供的代码的结果：

## 类方法

© 2000-2025, MetaQuotes Ltd.

| 创建 | |
|---|---|
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 添加 | |
| AddItem | 添加新项目 |
| 只读数据 | |
| 数值 | 获取控件关联的值。 |
| 依赖控件 | |
| CreateButton | 创建新的 CRadioButton 项目。 |
| 依赖控件事件处理器 | |
| OnVScrollShow | VScroll依赖控件的"显示"事件处理器（虚函数） |
| OnVScrollHide | VScroll依赖控件的"隐藏"事件处理器（虚函数） |
| OnScrollLineDown | VScroll依赖控件的"滚动条向下"事件处理器（虚函数） |
| OnScrollLineUp | VScroll依赖控件的"滚动条向上"事件处理器（虚函数） |
| OnChangeItem | "项目变化"事件处理器（虚函数） |
| 重绘 | |
| 重绘 | 重绘全组项目 |
| RowState | 设置指定项目的状态 |
| Select | 选择当前项目 |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide

方法继承自类CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

© 2000-2025, MetaQuotes Ltd.

创建单选键组面板的示例：

```
//+------------------------------------------------------------------+
//|                                          ControlsRadioGroup.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CRadioGroup"
#include <Controls\Dialog.mqh>
#include <Controls\RadioGroup.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CRadioGroup       m_radio_group;                  // CRadioGroup object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
```

© 2000-2025, MetaQuotes Ltd.

```
protected:
   //--- create dependent controls
   bool              CreateRadioGroup(void);
   //--- handlers of the dependent controls events
   void              OnChangeRadioGroup(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_radio_group,OnChangeRadioGroup)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateRadioGroup())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "RadioGroup" element                                  |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateRadioGroup(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y);
```

© 2000-2025, MetaQuotes Ltd.

```
   int x2=x1+GROUP_WIDTH;
   int y2=y1+RADIO_HEIGHT;
//--- create
   if(!m_radio_group.Create(m_chart_id,m_name+"RadioGroup",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_radio_group))
      return(false);
//--- fill out with strings
   for(int i=0;i<3;i++)
      if(!m_radio_group.AddItem("Item "+IntegerToString(i),1<<i))
         return(false);
   m_radio_group.Value(1<<2);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeRadioGroup(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
```

© 2000-2025, MetaQuotes Ltd.

```
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CRadioGroup 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# AddItem

添加新项目至控件。

```
bool  AddItem(
   const string  item,     // 文本
   const long    value=0   // 值
   )
```

参数

*item*

[输入] 文本。

*value=0*

[输入] 长整数 类型值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Value

获取控件关联的值。

```
long  Value()
```

返回值

控件关联的值。

注释

该值依赖 CRadioGroup 控件中的所有 CRadioButton 项目状态。

© 2000-2025, MetaQuotes Ltd.

# CreateButton

在指定索引处创建新的 CRadioButton 类实例。

```
bool  CreateButton(
   const int  index      // 索引
   )
```

参数

*index*

[输入] 在 CRadioGroup 中的新项目索引值。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollShow

控件的"垂直滚动显示"(垂直滚动条显示)事件处理器。

```
virtual bool  OnVScrollShow()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnVScrollHide

控件的"垂直滚动隐藏"(垂直滚动条隐藏)事件处理器。

```
virtual bool  OnVScrollHide()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineDown

控件的"滚动条向下"(垂直滚动条向下)事件处理器。

```
virtual bool  OnScrollLineDown()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnScrollLineUp

控件的"滚动条向上"(垂直滚动条向上)事件处理器。

```
virtual bool  OnScrollLineUp()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangeItem

控件的"项目改变"(项目变化)事件处理器。

```
virtual bool  OnChangeItem(
   const int  index      // 索引
   )
```

参数

*index*

[输入] 改变项目的索引.

返回值

true 如果事件已经被处理, 否则 false。

© 2000-2025, MetaQuotes Ltd.

# Redraw

重绘控件

```
bool  Redraw()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# RowState

设置指定项目的状态。

```
bool  RowState(
   const int   index,      // 项目索引
   const bool  select      // 状态
   )
```

参数

*index*

[输入] 即将改变的项目索引。

*select*

[输入] 新状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Select

选择当前项目。

```
void  Select(
   const int  index      // 索引
   )
```

参数

*index*

[输入] 选择的项目索引。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CSpinEdit

CSpinEdit 是 SpinEdit 复合控件之中的类(含依赖控件)。

## 描述

CSpinEdit 类的目的在于创建可编辑整数类型值的控件。它在按动向上按钮时自动递增数据，但若按动向下按钮则递减。

## 声明

```
class CSpinEdit : public CWndContainer
```

## 标称库文件

```
#include <Controls\SpinEdit.mqh>
```

## 继承体系

COBject
CWnd
CWndContainer
CSpinEdit

下面提供的代码的结果：

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| MinValue | 获取/设置最小允许值 |
| MaxValue | 获取/设置最大允许值 |
| 状态 | |
| 数值 | 获取/设置当前值 |
| 依赖控件 | |
| CreateEdit | 创建依赖控件（编辑） |
| CreateInc | 创建依赖控件（递增按钮） |
| CreateDec | 创建依赖控件（递减按钮） |
| 依赖控件事件处理器 | |
| OnClickInc | "点击递增"事件处理器（虚函数） |
| OnClickDec | "点击递减"事件处理器（虚函数） |
| 内部事件处理器 | |
| OnChangeValue | "数值变化"事件处理器（虚函数） |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

创建SpinEdit控制面板的示例：

```
//+------------------------------------------------------------------+
//|                                            ControlsSpinEdit.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
```

© 2000-2025, MetaQuotes Ltd.

```
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CSpinEdit"
#include <Controls\Dialog.mqh>
#include <Controls\SpinEdit.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CSpinEdit         m_spin_edit;                    // CSpinEdit object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateSpinEdit(void);
   //--- handlers of the dependent controls events
   void              OnChangeSpinEdit(void);
  };
//+------------------------------------------------------------------+
```

© 2000-2025, MetaQuotes Ltd.

```
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
   ON_EVENT(ON_CHANGE,m_spin_edit,OnChangeSpinEdit)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateSpinEdit())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "SpinEdit" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateSpinEdit(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+(BUTTON_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_spin_edit.Create(m_chart_id,m_name+"SpinEdit",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_spin_edit))
      return(false);
   m_spin_edit.MinValue(10);
   m_spin_edit.MaxValue(100);
   m_spin_edit.Value(50);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
```

© 2000-2025, MetaQuotes Ltd.

```
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeSpinEdit(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CSpinEdit 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## MinValue (Get方法)

获取控件的"最大小值"属性值 (最小允许值)。

```
int  MinValue()  const
```

返回值

控件的"最小值"属性值 。

## MinValue (Set方法)

设置控件的"最小值"属性值 (最小允许值)。

```
void  MinValue(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"最小位置"属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## MaxValue (Get方法)

获取控件的"最大值"属性值 (最大允许值)。

```
int  MaxValue()  const
```

返回值

控件的"最大值"属性值 。

## MaxValue (Set方法)

设置控件的"最大值"属性值 (最大允许值)。

```
void  MaxValue(
   const int  value      // 新值
   )
```

参数

*value*

[输入] 新的"最大值"属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

## Value (Get方法)

获取控件的"值"属性(当前值)。

```
int  Value()  const
```

返回值

"值"属性。

## Value (Set方法)

设置控件的"值"属性(当前值)。

```
void  Value(
   const int  value      // 值
   )
```

参数

*value*

[输入] 新的"值"属性值。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# CreateEdit

创建依赖控件 (CEdit)。

```
virtual bool  CreateEdit()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateInc

创建依赖控件 (递增按钮)。

```
virtual bool  CreateInc()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateDec

创建依赖控件（递减按钮）。

```
virtual bool  CreateDec()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickInc

控件的"点击递增"(鼠标在递增按钮上点击)事件处理器。

```
virtual bool  OnClickInc()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickDec

控件的"点击递减"(鼠标在递减按钮上点击)事件处理器。

```
virtual bool  OnClickDec()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnChangeValue

控件的"数值变化"事件处理器。

```
virtual bool  OnChangeValue()
```

返回值

true 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CDialog

CDialog 是 Dialog 复合控件之中的类。

## 描述

CDialog 类的意图是将 MQL5 程序内部不同组的功能控件结合到一起。

## 声明

```
class CDialog : public CWndContainer
```

## 标称库文件

```
#include <Controls\Dialog.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CDialog

直接分支

CAppDialog

## 类方法

| 创建 | |
|---|---|
| 创建 | 创建控件 |
| 图表事件处理器 | |
| OnEvent | 所有图表事件的句柄 |
| 属性 | |
| Caption | 获取/设置 "标题" 属性的值 |
| 添加 | |
| 添加 | 添加控件至客户区域 |
| 依赖控件 | |
| CreateWhiteBorder | 创建依赖控件 (白边) |
| CreateBackground | 创建依赖控件 (背景) |
| CreateCaption | 创建依赖控件 (标题) |
| CreateButtonClose | 创建依赖控件 (关闭按钮) |
| CreateClientArea | 创建依赖控件 (客户区域) |

© 2000-2025, MetaQuotes Ltd.

| | |
|---|---|
| 创建 | |
| 依赖控件事件处理器 | |
| OnClickCaption | "点击标题"事件处理器 |
| OnClickButtonClose | "点击按钮关闭"事件处理器（虚函数） |
| 访问客户区域 | |
| ClientAreaVisible | 设置客户区域是否可见的指示值 |
| ClientAreaLeft | 获取客户区域控件左上角的 X 坐标 |
| ClientAreaTop | 获取客户区域控件左上角的 Y 坐标 |
| ClientAreaRight | 获取客户区域控件右下角的 X 坐标 |
| ClientAreaBottom | 获取客户区域控件右下角的 Y 坐标 |
| ClientAreaWidth | 获取客户区域的宽度 |
| ClientAreaHeight | 获取客户区域的高度 |
| 拖动事件处理器 | |
| OnDialogDragStart | "对话框拖动开始"事件处理器（虚函数） |
| OnDialogDragProcess | "对话框拖动过程"事件处理器（虚函数） |
| OnDialogDragEnd | "对话框拖动结束"事件处理器（虚函数） |

方法继承自类CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CDialog 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

## Caption (Get方法)

获取CDialog控件的"标题"属性。

```
string  MinValue()  const
```

返回值

"标题"属性。

## Caption (Set方法)

设置CDialog控件的"标题"属性。

```
bool  Caption(
   const string  text      // 文本
   )
```

参数

*text*

[输入]"标题"属性的新值。

返回值

true如果成功，否则false。

© 2000-2025, MetaQuotes Ltd.

## Add

通过指针添加控件至客户区域。

```
bool  Add(
   CWnd   *control,         // 指针
   )
```

参数

*control*

[输入] 控件指针。

返回值

true 如果成功，否则 false。

## 添加

通过引用添加控件至客户区域。

```
bool  Add(
   CWnd   &control,         // 引用
   )
```

参数

*control*

[输入] 控件引用。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateWhiteBorder

创建依赖控件（白边）

```
virtual bool  CreateWhiteBorder()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateBackground

创建依赖控件（背景）。

```
virtual bool  CreateBackground()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateCaption

创建依赖控件（标题）。

```
virtual bool  CreateCaption()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateButtonClose

创建依赖控件（关闭按钮）

```
virtual bool  CreateButtonClose()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateClientArea

创建依赖控件（客户区域）

```
virtual bool  CreateClientArea()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickCaption

控件的"点击标题"事件处理器。。

```
virtual bool  OnClickCaption()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# OnClickButtonClose

控件的"点击按钮关闭"事件处理器。

```
virtual bool  OnClickButtonClose()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaVisible

设置客户区域是否可见的指示值。

```
bool  ClientAreaVisible(
   const bool  visible      // 可见性标志
   )
```

参数

*visible*

[输入] 可见性标志。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaLeft

获取客户区域控件左上角的 X 坐标。

```
int  ClientAreaLeft()
```

返回值

客户区域控件 左上角的 X 坐标。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaTop

获取客户区域控件左上角的 Y 坐标。

```
int  ClientAreaTop()
```

返回值

客户区域控件 左上角的 Y 坐标。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaRight

获取客户区域控件右下角的 X 坐标。

```
int  ClientAreaTop()
```

返回值

客户区域控件右下角的 X 坐标。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaBottom

获取客户区域控件右下角的 Y 坐标。

```
int  ClientAreaBottom()
```

返回值

客户区域控件右下角的 Y 坐标。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaWidth

获取客户区域控件宽度。

```
int  ClientAreaWidth()
```

返回值

客户区域控件宽度。

© 2000-2025, MetaQuotes Ltd.

# ClientAreaHeight

获取客户区域控件高度。

```
int  ClientAreaHeight()
```

返回值

客户区域控件高度。

© 2000-2025, MetaQuotes Ltd.

# OnDialogDragStart

控件的"对话框拖开始"事件处理器。

```
virtual bool  OnDialogDragStart()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当拖动控件开始时"对话框拖动开始"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnDialogDragProcess

控件的"对话框拖动过程"事件处理器。

```
virtual bool  OnDialogDragProcess()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当拖动控件时"对话框拖动过程"事件触发。

© 2000-2025, MetaQuotes Ltd.

# OnDialogDragEnd

控件的 "对话框拖动结束" 事件处理器。

```
virtual bool  OnDialogDragEnd()
```

返回值

true 如果事件已经被处理，否则 false。

注释

当拖动控件结束时 "对话框拖动结束" 事件触发。

© 2000-2025, MetaQuotes Ltd.

# CAppDialog

CAppDialog 是应用对话框复合控件之中的类(含依赖控件)。

## 描述

CAppDialog 类的意图是将 MQL5 程序内部不同组的功能控件结合到一起。

## 声明

```
class CAppDialog : public CDialog
```

## 标称库文件

```
#include <Controls\Dialog.mqh>
```

## 继承体系

CObject
  CWnd
    CWndContainer
      CDialog
        CAppDialog

## 类方法

| | |
|---|---|
| 创建于销毁 | |
| 创建 | 创建控件 |
| 销毁 | 销毁控件 |
| 事件处理 | |
| OnEvent | 所有图表事件的句柄 |
| 运行 | |
| 运行 | 运行控件 |
| 图表事件处理 | |
| ChartEvent | 所有图表事件的句柄 |
| 设置 | |
| Minimized | 设置控件是否为最小化的指示值 |
| 保存/加载 | |
| IniFileSave | 保存控件状态至文件 |
| IniFileLoad | 从文件加载控件状态 |
| IniFileName | 设置加载/保存控件状态的文件名 |

© 2000-2025, MetaQuotes Ltd.

| 创建于销毁 | |
|---|---|
| IniFileExt | 设置加载/保存控件状态的文件扩展名 |
| 初始化 | |
| CreateCommon | 一般初始化方法 |
| CreateExpert | 配合智能交易工作的初始化方法 |
| CreateIndicator | 配合指标工作的初始化方法 |
| 依赖控件 | |
| CreateButtonMinMax | 创建依赖控件（最小化/最大化按钮） |
| 依赖控件事件处理器 | |
| OnClickButtonClose | "点击按钮关闭"事件处理器（虚函数） |
| OnClickButtonMinMax | "点击按钮最小最大"事件处理器（虚函数） |
| 外部事件 | |
| OnAnotherApplicationClose | 外部事件处理器（虚函数） |
| 方法 | |
| Rebound | 设置 CRect 类的新坐标 |
| Minimize | 显示控件的最小化状态 |
| Maximize | 显示控件的最大化状态 |
| CreateInstanceId | 为控件名称创建独有 id |
| ProgramName | 获取 MQL5 程序名称 |
| SubwinOff | 获取控件子窗口的 Y 偏移 |

方法继承自类 CObject

Prev, Prev, Next, Next, Type, Compare

方法继承自类 CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

方法继承自类 CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

方法继承自类 CDialog

Caption, Caption, Add, Add

© 2000-2025, MetaQuotes Ltd.

© 2000-2025, MetaQuotes Ltd.

# Create

创建新的 CAppDialog 控件。

```
virtual bool  Create(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Destroy

CAppDialog控制去初始化方法

```
virtual void  Destroy(
   const int  reason=REASON_PROGRAM    // 原因代码
)
```

参数

*reason*

[in] 去初始化原因代码。REASON_PROGRAM被默认设置。

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnEvent

图表事件处理器。

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Run

运行控件。

```
bool  Run()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# ChartEvent

图表事件处理器。

```
virtual bool  ChartEvent(
   const int      id,          // ID
   const long&    lparam,      // 长整数型事件参数
   const double&  dparam,      // 双精度型事件参数
   const string&  sparam       // 字符串型事件参数
   )
```

参数

*id*

[输入] 事件 ID。

*lparam*

[输入] 事件参数 长整数 型，通过引用传递。

*dparam*

[输入] 事件参数 双精度 型，通过引用传递。

*sparam*

[输入] 事件参数 字符串 型，通过引用传递。

返回值

true - 如果事件已经被处理，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Minimized

设置控件"最小化"属性的值。

```
bool  Minimized(
   const bool  flag      // 状态
   )
```

参数

*flag*

[输入] 新状态。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# IniFileSave

保存控件状态至文件。

```
void  IniFileSave()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# IniFileLoad

从文件加载控件状态。

```
void  IniFileLoad()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# IniFileName

设置加载/保存控件状态的文件名。

```
virtual string  IniFileName()  const
```

返回值

文件名，用于加载/保存控件状态。

注释

文件名包括智能交易/指标名称以及工作品名，即 MQL5 程序启动时对应的品种。

© 2000-2025, MetaQuotes Ltd.

# IniFileExt

设置加载/保存控件状态的文件扩展名。

```
virtual string  IniFileExt()  const
```

返回值

文件扩展名，用于加载/保存控件状态。

© 2000-2025, MetaQuotes Ltd.

# CreateCommon

一般初始化方法。

```
bool  CreateCommon(
   const long     chart,       // 图表 ID
   const string   name,        // 名称
   const int      subwin,      // 图表子窗口
   )
```

参数

*chart*

[输入] 图表 ID。

*name*

[输入] 控件的独有名称。

*subwin*

[输入] 图表子窗口。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateExpert

配合智能交易工作的初始化方法。

```
bool  CreateExpert(
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateIndicator

配合指标工作的初始化方法。

```
bool  CreateIndicator(
   const int      x1,          // x1 坐标
   const int      y1,          // y1 坐标
   const int      x2,          // x2 坐标
   const int      y2           // y2 坐标
   )
```

参数

*x1*

[输入] 左上角的 X 坐标。

*y1*

[输入] 左上角的 Y 坐标。

*x2*

[输入] 右下角的 X 坐标。

*y2*

[输入] 右下角的 Y 坐标。

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateButtonMinMax

创建依赖控件（最小化/最大化按钮）。

```
virtual void  CreateButtonMinMax()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnClickButtonClose

控件的"点击按钮关闭"(鼠标点击关闭按钮)事件处理器。

```
virtual void  OnClickButtonClose()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnClickButtonMinMax

控件的"点击按钮最小最大"(鼠标点击最小化/最大化按钮)事件处理器。

```
virtual void  OnClickButtonClose()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# OnAnotherApplicationClose

外部状态事件处理器。

```
virtual void  OnAnotherApplicationClose()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# Rebound

设置 CRect 类的新坐标。

```
bool  Rebound(
   const & CRect  rect      // CRect 类
   )
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Minimize

显示控件的最小化状态。

```
virtual void  Minimize()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# Maximize

显示控件的最大化（复原）状态。

```
virtual void  Maximize()
```

返回值

true 如果成功，否则 false。

© 2000-2025, MetaQuotes Ltd.

# CreateInstanceId

为控件名称创建独有 id。

```
string  CreateInstanceId()
```

返回值

对象名称前缀。

© 2000-2025, MetaQuotes Ltd.

# ProgramName

获取 MQL5 程序名称。

```
string  ProgramName()
```

返回值

MQL5 程序名称。

© 2000-2025, MetaQuotes Ltd.

# SubwinOff

获取控件子窗口的 Y 偏移。

```
void  SubwinOff()
```

返回值

无。

© 2000-2025, MetaQuotes Ltd.

# 从 MQL4 到 MQL5

MQL5是前任MQL4语言的发展，其中编辑进去许多指标，脚本和EA交易。尽管事实上，新的程序语言最大化的与前一代相匹配，但在这些语言之间，仍有区别，当转换程序时，这些区别应该标注出来。

对于了解MQL4的程序员来说，新的MQL5语言，该章节涵盖了促进代码改编进化的所有信息。

首先应该标明的是：

- 新语言不包括函数start()，init()和deinit()；
- 指标缓冲区数量不限制；
- 在下载EA程序后（或者其他MQL5程序），dll立即被加载；
- 检测缩写的逻辑条件；
- 当数组超过限制时，结束当前操作（紧急的-错误输出）；
- 像C++一样运算符优先；
- 该语言提供隐式类型（甚至从字符串到数字）；
- 局部变量不能自动初始化（除了字符串）；
- 普通本地数组自动删除。

## 特殊函数 init, start and deinit

MQL4语言只包括3个预定函数可以操作指标，脚本或者EA交易（不包括files *.mqh和数据库文件）。在MQL5中没有那样的函数，但有与之相类似的函数，图表表明了近似函数。

| MQL4 | MQL5 |
|---|---|
| init | OnInit |
| start | OnStart |
| deinit | OnDeinit |

在MQL4中，函数 OnInit 和 OnDeinit 在 init 和 deinit 程序中执行了相同的任务-他们是为本地代码设计的，一定在MQL5程序的初始化过程中执行，可以因此重命名函数，或者以他们本身的形式呈现，但是在类似位置这些函数可以添加调用。

示例：

```
void OnInit()
  {
//--- 调用函数去初始化
   init();
  }
void OnDeinit(const int reason)
  {
//--- 调用无法初始化函数
   deinit();
//---
  }
```

© 2000-2025, MetaQuotes Ltd.

开始函数只在脚本中被OnStart代替，在EA交易和指标中应该分别重命名成OnTick和OnCalculate。在MQL5程序操作过程中，这些代码可以执行，并保存3个函数中：

| mql5程序 | 主函数 |
|---|---|
| 脚本 | OnStart |
| 指标 | OnCalculate |
| EA交易 | OnTick |

如果指标或者代码不包括主函数，或者函数名称与要求的不相符，该函数的调用就不能执行。这表示了，如果脚本的资源代码不包括OnStart，该代码会以一个EA交易编辑。

如果指标代码不包括OnCalculate函数，该指标的编辑不能完成。

## 预定义变量

在MQL5中有诸如Ask, Bid, Bars的预定义变量，变量点和数字的拼写有稍许不同：

| MQL4 | MQL5 |
|---|---|
| Digits | _Digits |
| Point | _Point |
| | _LastError |
| | _Period |
| | _Symbol |
| | _StopFlag |
| | _UninitReason |

## 访问时间序列

MQL5中没有像Open[], High[], Low[], Close[], Volume[] and Time[]这样预定义的时间序列。时间序列必要的深度能用相关访问时间序列函数来设定。

## EA交易

MQL5的EA交易不需要强制存在处理新订单号收据事件的函数-OnTick，如在MQL4中一样（当接收新订单时执行MQL4的启动函数），因为在MQL5中EA交易可以包含几种类型的预定义处理器函数。

- OnTick -新订单号收据；
- OnTimer 时间数据；
- OnTrade -交易事件；
- OnChartEvent -键盘鼠标输入事件，图解物件移动事件，完成文本编辑事件输入LabelEdit物件领域中的；

© 2000-2025, MetaQuotes Ltd.

- OnBookEvent -市场深度状态更改事件。

## 自定义指标

在MQL4中，指标缓冲区的数量是有限的，不能超过8。而在MQL5中则没有这个限制，但是要记得每个指标缓冲区需要在程序端分配部分内存，所以，这个新功能也不能滥用。

MQL4只提供了6种自定义指标绘图；而MQL5现在提供了18种绘画类型 。绘画类型的名称不变，但是指标的图解表示意义却显著变化。

指标缓冲区中索引趋势也并不相同。默认情况下，在MQL5中，所有指标缓冲区都有常用数组行为，例如0索引元素是历史记录中最古老的一个，随着指数增加，数据也从旧的换成新的。

受MQL4保护的工作自定义指标的唯一函数是SetIndexBuffer 。但是调用改变；现在应该指定存储在数组中的数据类型，连接指标缓冲区。

自定义指标属性也已更改并发展。添加了访问时间序列的新函数，所以全部算法需要重新审议。

## 图解物件

在MQL5中图解物件的数量显著增加。此外，图解物件现在可以及时置于时间表的图表中且可精确到秒-现在物件定位点不会四舍五入到当前价格图表开盘时间柱上。对于箭头物件，可以指定文本和标签绑定方式 ，并且对于标签，可以设置按钮，图表，位图标签和编辑定位定位物件的图表角 。

© 2000-2025, MetaQuotes Ltd.

# MQL5 函数列表

MQL5 所有函数都是按照字母顺序排列的。

| 函数 | 功能 | 章节 |
|---|---|---|
| AccountInfoDouble | 返回相应账户属性的双精度值 | 账户信息 |
| AccountInfoInteger | 返回相应账户属性的整数类型值（布尔，整型或者长整型） | 账户信息 |
| AccountInfoString | 返回相应账户属性的字符串类型值 | 账户信息 |
| acos | 返回x弧度的反余弦 | 数学函数 |
| Alert | 在独立窗口中显示消息 | 普通函数 |
| ArrayBsearch | 在第一函数维度索引第一组发现的元素 | 数组函数 |
| ArrayCompare | 返回比较简单类型或无复杂对象的自定义结构的两种数组的结果 | 数组函数 |
| ArrayCopy | 复制一组数组到另一组 | 数组函数 |
| ArrayFill | 以指定值填充数组 | 数组函数 |
| ArrayFree | 解放任意动态数组的缓冲区并建立0维度大小 | 数组函数 |
| ArrayGetAsSeries | 检测数组标引的导向 | 数组函数 |
| ArrayInitialize | 给数字数组所有元素一个单独的值 | 数组函数 |
| ArrayIsDynamic | 检测是否数组是动态的 | 数组函数 |
| ArrayIsSeries | 检测是否数组是时序列 | 数组函数 |
| ArrayMaximum | 检测元素的最大值 | 数组函数 |
| ArrayMinimum | 检测元素的最小值 | 数组函数 |
| ArrayRange | 在数组的指定维度返回元素数量 | 数组函数 |
| ArrayResize | 在数组的第一维尺寸设定新大小 | 数组函数 |
| ArraySetAsSeries | 指明数组标引的导向 | 数组函数 |
| ArraySize | 返回数组中元素的数量 | 数组函数 |
| ArraySort | 通过第一维度排列数值数组 | 数组函数 |
| ArrayPrint | 将简单类型或简单结构的数组打印到日志 | 数组函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| ArrayInsert | 从指定索引开始，将来自源数组的指定元素数量插入到接收数组 | 数组函数 |
| ArrayRemove | 从指定索引开始移除数组中指定的元素数量 | 数组函数 |
| ArrayReverse | 从指定索引开始反转数组中指定的元素数量 | 数组函数 |
| ArraySwap | 交换相同类型两个动态数组的内容 | 数组函数 |
| asin | 返回x弧度的反正弦 | 数学函数 |
| atan | 返回x弧度的反正切 | 数学函数 |
| Bars | 返回规定交易品种和周期的历史字节数量 | 时间序列和指标访问 |
| BarsCalculated | 错误情况下（还没有计算数据）返回指标缓冲区计算的数据量或者-1 | 时间序列和指标访问 |
| CalendarCountryById | 根据ID获得国家描述 | 经济日历函数 |
| CalendarEventById | 根据ID获得事件描述 | 经济日历函数 |
| CalendarValueById | 根据ID获得事件值描述 | 经济日历函数 |
| CalendarCountries | 获得日历中可用的国家名称数组 | 经济日历函数 |
| CalendarEventByCountry | 根据指定国家代码获得日历中可用的所有事件描述数组 | 经济日历函数 |
| CalendarEventByCurrency | 根据指定货币获得日历中可用的所有事件描述数组 | 经济日历函数 |
| CalendarValueHistoryByEvent | 根据事件ID获得指定时期的所有事件值数组 | 经济日历函数 |
| CalendarValueHistory | 通过根据国家和/或货币进行排序的能力，获得指定时期的所有事件值数组 | 经济日历函数 |
| CalendarValueLastByEvent | 通过指定change_id，从日历数据库状态根据ID获得事件值数组 | 经济日历函数 |
| CalendarValueLast | 通过指定change_id和根据国家和/或货币进行排序的能力，从日历数据库状态获得所有事件值数组 | 经济日历函数 |
| ceil | 从上面返回最靠近的整数数值 | 数学函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| CharArrayToString | 转换符号代码 (ansi)到单符号数组 | 函数转换 |
| ChartApplyTemplate | 在图表中应用指定文件中的特定模板。 | 图表操作 |
| ChartClose | 关闭指定图表。 | 图表操作 |
| ChartFirst | 返回客户端第一图表的ID。 | 图表操作 |
| ChartGetDouble | 返回指定图表的双精度值属性。 | 图表操作 |
| ChartGetInteger | 返回指定图表的整数值属性。 | 图表操作 |
| ChartGetString | 返回指定图表的字符串属性。 | 图表操作 |
| ChartID | 返回当前图表的ID。 | 图表操作 |
| ChartIndicatorAdd | 用指定处理程序在指定图表窗口填加指标。 | 图表操作 |
| ChartIndicatorDelete | 从指定图表窗口移除一个指定名称的指标。 | 图表操作 |
| ChartIndicatorGet | 返回在指定图表窗口指定缩略名的指标句柄。 | 图表操作 |
| ChartIndicatorName | 返回指定图表窗口的指标列表的指标缩略名数量。 | 图表操作 |
| ChartIndicatorsTotal | 返回应用到指定图表窗口全部指标数量。 | 图表操作 |
| ChartNavigate | 通过指定的关系柱完成指定图表到图表中的指定位置的转换。 | 图表操作 |
| ChartNext | 返回指定图表旁边图表的ID。 | 图表操作 |
| ChartOpen | 打开指定交易品种和周期的新图表。 | 图表操作 |
| CharToString | 转换符号代码到字符串 | 函数转换 |
| ChartPeriod | 返回指定图表的周期值。 | 图表操作 |
| ChartPriceOnDropped | 返回图表点EA交易或者脚本下跌到的价格坐标。 | 图表操作 |
| ChartRedraw | 调用指定图表的被怕重画。 | 图表操作 |
| ChartSaveTemplate | 以指定名称在模板中保存当前图表设置 | 图表操作 |
| ChartScreenShot | 以gif,png或bmp格式提供图表当前状态屏幕截图。 | 图表操作 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| ChartSetDouble | 设置指定图表相关属性的双精度值。 | 图表操作 |
| ChartSetInteger | 设置指定图表相关属性的整数值（日期时间，整型，颜色，布尔或者字符型）。 | 图表操作 |
| ChartSetString | 设置指定图表相关属性的字符串值。 | 图表操作 |
| ChartSetSymbolPeriod | 改变交易品种值和指定图表周期。 | 图表操作 |
| ChartSymbol | 返回指定图表的交易品种名称。 | 图表操作 |
| ChartTimeOnDropped | 返回图表点EA交易或者脚本下跌到的时间坐标。 | 图表操作 |
| ChartTimePriceToXY | 从表现时间/价格的图表坐标转变到X和Y轴 | 图表操作 |
| ChartWindowFind | 返回绘画指标的子窗口数量。 | 图表操作 |
| ChartWindowOnDropped | 返回图表子窗口EA交易或者脚本下跌到的数（指数）。 | 图表操作 |
| ChartXOnDropped | 返回图表点EA交易或者脚本下跌到的X坐标。 | 图表操作 |
| ChartXYToTimePrice | 转变图表的X和Y轴到时间和价格值 | 图表操作 |
| ChartYOnDropped | 返回图表点EA交易或者脚本下跌到的Y坐标。 | 图表操作 |
| CheckPointer | 返回目标指针类型 | 普通函数 |
| CLBufferCreate | 创建OpenCL缓冲 | 使用OpenCL工作 |
| CLBufferFree | 删除OpenCL缓冲 | 使用OpenCL工作 |
| CLBufferRead | 读取OpenCL缓冲到数组 | 使用OpenCL工作 |
| CLBufferWrite | 写入数组到OpenCL缓冲 | 使用OpenCL工作 |
| CLContextCreate | 创建OpenCL背景 | 使用OpenCL工作 |
| CLContextFree | 移除OpenCL背景 | 使用OpenCL工作 |
| CLExecute | 运行OpenCL程序 | 使用OpenCL工作 |
| CLGetDeviceInfo | 从OpenCL驱动程序接收设备属性 | 使用OpenCL工作 |
| CLGetInfoInteger | 为OpenCL对象或设备返回一个整数属性值 | 使用OpenCL工作 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| CLHandleType | 返回OpenCL处理类型作为ENUM_OPENCL_HANDLE_TYPE 详表中的某个值. | 使用OpenCL工作 |
| CLKernelCreate | 创建OpenCL开始函数 | 使用OpenCL工作 |
| CLKernelFree | 移除OpenCL开始函数 | 使用OpenCL工作 |
| CLProgramCreate | 从源代码创建OpenCL程序 | 使用OpenCL工作 |
| CLProgramFree | 移除OpenCL程序 | 使用OpenCL工作 |
| CLSetKernelArg | 为OpenCL函数设置一个参数 | 使用OpenCL工作 |
| CLSetKernelArgMem | 设置OpenCL缓冲作为OpenCL函数的参数 | 使用OpenCL工作 |
| ColorToARGB | 转换color类型到uint类型，接收ARGB颜色显示。 | 函数转换 |
| ColorToString | 转换颜色值到"R,G,B"形式的字符串。 | 函数转换 |
| Comment | 图表左上角输出注解文本 | 普通函数 |
| CopyBuffer | 从指定指标获得特定缓冲区数据到数组中。 | 时间序列和指标访问 |
| CopyClose | 将指定交易品种收盘价格柱的历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyHigh | 将指定交易品种最大价格柱的历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyLow | 将指定交易品种最小价格柱的历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyOpen | 将指定交易品种 开盘价格柱的历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyRates | 将指定交易品种 比率结构的历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyRealVolume | 指定交易品种交易量历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopySpread | 指定交易品种点差历史数据和周期放在数组中 | 时间序列和指标访问 |
| CopyTicks | 获得程序端为当前工作期积累的报价并写入数组 | 时间序列和指标访问 |
| CopyTickVolume | 指定交易品种订单交易量历史数据和周期放在数组中 | 时间序列和指标访问 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| CopyTime | 将指定交易品种 开盘时间柱的历史数据和周期放在数组中 | 时间序列和指标访问 |
| cos | 返回数字余弦 | 数学函数 |
| CryptDecode | 执行数组数据逆转换 | 普通函数 |
| CryptEncode | 通过指定方式变换来自数组的数据 | 普通函数 |
| CustomSymbolCreate | 在指定组以指定名称创建一个自定义交易品种 | 自定义交易品种 |
| CustomSymbolDelete | 删除指定名称的自定义交易品种 | 自定义交易品种 |
| CustomSymbolSetInteger | 为自定义交易品种设置整型属性值 | 自定义交易品种 |
| CustomSymbolSetDouble | 为自定义交易品种设置真实型属性值 | 自定义交易品种 |
| CustomSymbolSetString | 为自定义交易品种设置字符串类型属性值 | 自定义交易品种 |
| CustomSymbolSetMarginRate | 按照订单类型和方向，为自定义交易品种设置预付款比率 | 自定义交易品种 |
| CustomSymbolSetSessionQuote | 为指定交易品种和工作日设置指定报价时段的起止时间 | 自定义交易品种 |
| CustomSymbolSetSessionTrade | 为指定交易品种和工作日设置指定交易时段的起止时间 | 自定义交易品种 |
| CustomRatesDelete | 删除指定时间间隔内自定义交易品种价格历史的全部柱形图 | 自定义交易品种 |
| CustomRatesReplace | 以MqlRates类型数组数据完全替换指定时间间隔内自定义交易品种的价格历史 | 自定义交易品种 |
| CustomRatesUpdate | 将丢失的柱形图添加到自定义交易品种历史并用MqlRates类型数组数据替换现有数据 | 自定义交易品种 |
| CustomTicksAdd | 添加MqlTick类型的数组数据到自定义交易品种的价格历史。自定义交易品种一定要在市场报价窗口来选择 | 自定义交易品种 |
| CustomTicksDelete | 删除指定时间间隔内自定义交易品种价格历史的全部报价 | 自定义交易品种 |
| CustomTicksReplace | 以MqlTick类型数组数据完全替换指定时间间隔内自定义交易 | 自定义交易品种 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| | 品种的价格历史 | |
| CustomBookAdd | 为自定义交易品种传递市场深度的状态 | 自定义交易品种 |
| DatabaseOpen | 在指定文件中打开或创建数据库 | 使用数据库 |
| DatabaseClose | 关闭数据库 | 使用数据库 |
| DatabaseImport | 从文件导入数据到表格中 | 使用数据库 |
| DatabaseExport | 将表格或SQL请求执行结果到处到CSV文件 | 使用数据库 |
| DatabasePrint | 在专家日志中打印表格或SQL执行结果 | 使用数据库 |
| DatabaseTableExists | 检查数据库中是否存在表格 | 使用数据库 |
| DatabaseExecute | 执行对指定数据库的请求 | 使用数据库 |
| DatabasePrepare | 创建可使用DatabaseRead()执行的请求句柄 | 使用数据库 |
| DatabaseReset | 重置请求，比如调用DatabasePrepare()之后 | 使用数据库 |
| DatabaseBind | 在请求中设置一个参数值 | 使用数据库 |
| DatabaseBindArray | 将数组设置为参数值 | 使用数据库 |
| DatabaseRead | 作为请求结果，移到下一个条目 | 使用数据库 |
| DatabaseFinalize | 移除在DatabasePrepare()中创建的请求 | 使用数据库 |
| DatabaseTransactionBegin | 开始事务执行 | 使用数据库 |
| DatabaseTransactionCommit | 完成事务执行 | 使用数据库 |
| DatabaseTransactionRollback | 回滚事务 | 使用数据库 |
| DatabaseColumnsCount | 获取请求中的字段数 | 使用数据库 |
| DatabaseColumnName | 按索引获取字段名 | 使用数据库 |
| DatabaseColumnType | 按索引获取字段类型 | 使用数据库 |
| DatabaseColumnSize | 获取字段大小（以字节为单位） | 使用数据库 |
| DatabaseColumnText | 从当前记录中获取作为字符串的字段值 | 使用数据库 |
| DatabaseColumnInteger | 从当前记录中获取int类型的值 | 使用数据库 |
| DatabaseColumnLong | 从当前记录中获取long类型的值 | 使用数据库 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| DatabaseColumnDouble | 从当前记录中获取double类型的值 | 使用数据库 |
| DatabaseColumnBlob | 从当前记录中获取作为数组的字段值 | 使用数据库 |
| | | |
| DebugBreak | 排除故障中的程序断点 | 普通函数 |
| Digits | 返回决定当前图表交易品种价格值的精确性的小数位数 | 检测 |
| DoubleToString | 以规定的精确度转换数字值到文本行 | 函数转换 |
| DXContextCreate | 创建图形环境，以渲染指定大小的画帧 | 使用DirectX |
| DXContextSetSize | 更改在DXContextCreate()中创建的图形环境的画帧大小 | 使用DirectX |
| DXContextSetSize | 获取在DXContextCreate()中创建的图形环境的画帧大小 | 使用DirectX |
| DXContextClearColors | 为渲染缓冲区的所有像素设置指定的颜色 | 使用DirectX |
| DXContextClearDepth | 清除深度缓冲区 | 使用DirectX |
| DXContextGetColors | 从图形环境获取具有指定大小和偏移量的图像 | 使用DirectX |
| DXContextGetDepth | 获取渲染画帧的深度缓冲区 | 使用DirectX |
| DXBufferCreate | 根据数据数组创建指定类型的缓冲区 | 使用DirectX |
| DXTextureCreate | 从传递图像中剪切的指定大小的矩形中创建2D结构 | 使用DirectX |
| DXInputCreate | 创建着色器输入 | 使用DirectX |
| DXInputSet | 设置着色器输入 | 使用DirectX |
| DXShaderCreate | 创建指定类型的着色器 | 使用DirectX |
| DXShaderSetLayout | 设置顶点着色器的顶点布局 | 使用DirectX |
| DXShaderInputsSet | 设置着色器输入 | 使用DirectX |
| DXShaderTexturesSet | 设置着色器结构 | 使用DirectX |
| DXDraw | 渲染在DXBufferSet()中设置的顶点缓冲区的顶点 | 使用DirectX |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| DXDrawIndexed | 渲染由DXBufferSet()的索引缓冲区描述的图形原语 | 使用DirectX |
| DXPrimiveTopologySet | 设置使用DXDraw Indexed()渲染的原语类型 | 使用DirectX |
| DXBufferSet | 设置当前渲染的缓冲区 | 使用DirectX |
| DXShaderSet | 设置渲染的着色器 | 使用DirectX |
| DXHandleType | 返回句柄类型 | 使用DirectX |
| DXRelease | 释放句柄 | 使用DirectX |
| EnumToString | 将任何类型的枚举值转换为字符串 | 函数转换 |
| EventChartCustom | 为指定图表生成自定义事件 | 工作事件 |
| EventKillTimer | 依据当前图表定时器停止事件生成器 | 工作事件 |
| EventSetMillisecondTimer | 为当前图表启动周期少于1秒的高分辨率计时器的事件生成器 | 工作事件 |
| EventSetTimer | 用当前图表指定周期启动定时器事件生成器 | 工作事件 |
| exp | 返回数字指数 | 数学函数 |
| ExpertRemove | 停止智能交易并从图表中卸载 | 普通函数 |
| fabs | 返回指定数值的绝对值（绝对值） | 数学函数 |
| FileClose | 关闭之前打开的文件 | 文件函数 |
| FileCopy | 从本地或者共享文件夹复制原文件到另一个文件 | 文件函数 |
| FileDelete | 删除指定文件 | 文件函数 |
| FileFindClose | 结束搜索程序 | 文件函数 |
| FileFindFirst | 依照指定过滤器启动目录中的文件搜索 | 文件函数 |
| FileFindNext | 继续通过FileFindFirst()函数启动的搜索 | 文件函数 |
| FileFlush | 将输入/输出文件缓冲区的所有数据写入磁盘 | 文件函数 |
| FileGetInteger | 获得文件的整型属性 | 文件函数 |
| FileIsEnding | 读取过程中定义文件末端 | 文件函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| FileIsExist | 检测文件存在性 | 文件函数 |
| FileIsLineEnding | 读取过程中定义文本行末端 | 文件函数 |
| FileMove | 移动或者重命名文件 | 文件函数 |
| FileOpen | 打开指定名字和标记的文件 | 文件函数 |
| FileReadArray | 读取除了BIN类型文件字符串外的任何类型数组 | 文件函数 |
| FileReadBool | 读取CSV类型文件当前位置字符串直到定界符（或者直到文本行末端）和转换读取的字符串到布尔型值 | 文件函数 |
| FileReadDatetime | 读取CSV类型文件以"YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" or "HH:MI:SS"格式字符串并且转换它到日期时间值 | 文件函数 |
| FileReadDouble | 从文件指针当前位置读取双精度值 | 文件函数 |
| FileReadFloat | 从文件指针当前位置读取浮点值 | 文件函数 |
| FileReadInteger | 从文件指针当前位置读取整型，短整型或者字符型值 | 文件函数 |
| FileReadLong | 从文件指针当前位置读取一个长整型值 | 文件函数 |
| FileReadNumber | 从CSV类型文件读取当前位置字符串直到定界符（或者直到文本行末端）并且转换读取字符串到双精度值 | 文件函数 |
| FileReadString | 在文件中文件指针当前位置读取字符串 | 文件函数 |
| FileReadStruct | 文件指针当前位置读取二进制文件内容到作为参量传递的结构 | 文件函数 |
| FileSeek | 根据指定位置有关的指定二进制数移动文件指针位置 | 文件函数 |
| FileSize | 返回相应打开文件大小 | 文件函数 |
| FileTell | 返回相应打开文件的文件指针的当前位置 | 文件函数 |
| FileWrite | 向CSV或者TXT类型文件写入数据 | 文件函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| FileWriteArray | 写入字符串以外任何类型数组到BIN类型文件 | 文件函数 |
| FileWriteDouble | 从文件指针当前位置写入双精度型值到二进制文件 | 文件函数 |
| FileWriteFloat | 从文件指针当前位置写入浮点型值到二进制文件 | 文件函数 |
| FileWriteInteger | 从文件指针当前位置写入整型值到二进制文件 | 文件函数 |
| FileWriteLong | 从文件指针当前位置写入长整型值到二进制文件 | 文件函数 |
| FileWriteString | 从文件指针当前位置写入字符串参量值到BIN或者TXT文件 | 文件函数 |
| FileWriteStruct | 从文件指针当前位置写入作为参量传递的结构到二进制文件 | 文件函数 |
| floor | 从下面返回最靠近的整数数值 | 数学函数 |
| fmax | 返回两个数值的最大值 | 数学函数 |
| fmin | 返回两个数值的最小值 | 数学函数 |
| fmod | 两个数值相除后返回实余数 | 数学函数 |
| FolderClean | 删除指定文件夹中所有文件 | 文件函数 |
| FolderCreate | 在文件目录创建一个文件夹 | 文件函数 |
| FolderDelete | 清除选定目录。如果文件夹不是空的，那么无法清除。 | 文件函数 |
| FrameAdd | 添加数据帧 | 处理优化结果 |
| FrameFilter | 设置帧阅读过滤器并移动指针至起点 | 处理优化结果 |
| FrameFirst | 移动阅读帧的指针至起点并重置之前设置的过滤器 | 处理优化结果 |
| FrameInputs | 接收形成帧的 输入参数 | 处理优化结果 |
| FrameNext | 阅读帧并移动指针至下一个 | 处理优化结果 |
| GetLastError | 返回上一个错误 | 检测 |
| GetPointer | 返回对象 指针 | 普通函数 |
| GetTickCount | 从系统开始，返回已经过去的毫秒的数量 | 普通函数 |
| GlobalVariableCheck | 检测带有指定名的全局变量的存在性 | 程序端全局变量 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| GlobalVariableDel | 删除全局变量 | 程序端全局变量 |
| GlobalVariableGet | 返回全局变量值 | 程序端全局变量 |
| GlobalVariableName | 全局变量列表中依据序列号返回全局变量名 | 程序端全局变量 |
| GlobalVariablesDeleteAll | 删除指定前缀的全局变量 | 程序端全局变量 |
| GlobalVariableSet | 为全局变量设新值 | 程序端全局变量 |
| GlobalVariableSetOnCondition | 根据状态设置现存全局变量的新值 | 程序端全局变量 |
| GlobalVariablesFlush | 强制在磁盘保存全局变量内容 | 程序端全局变量 |
| GlobalVariablesTotal | 返回全局变量总数 | 程序端全局变量 |
| GlobalVariableTemp | 给全局变量设新值，只在程序端当前状态存在 | 程序端全局变量 |
| GlobalVariableTime | 返回上次访问全局变量的时间 | 程序端全局变量 |
| HistoryDealGetDouble | 返回历史记录中交易需求属性（双精度） | 交易函数 |
| HistoryDealGetInteger | 返回历史记录中交易需求属性（时间日期和整型） | 交易函数 |
| HistoryDealGetString | 返回历史记录中交易需求属性（字符串） | 交易函数 |
| HistoryDealGetTicket | 返回历史相关交易订单号 | 交易函数 |
| HistoryDealSelect | 历史记录中选择一个交易便于通过适当函数进一步调用它。 | 交易函数 |
| HistoryDealsTotal | 返回历史记录交易数 | 交易函数 |
| HistoryOrderGetDouble | 返回（双精度）历史记录中订单需求属性 | 交易函数 |
| HistoryOrderGetInteger | 返回（日期时间或者整型）历史记录中订单需求属性 | 交易函数 |
| HistoryOrderGetString | 返回（字符串）历史记录中订单需求属性 | 交易函数 |
| HistoryOrderGetTicket | 返回历史记录中相关订单的订单号 | 交易函数 |
| HistoryOrderSelect | 为下一步工作选择历史订单 | 交易函数 |
| HistoryOrdersTotal | 返回历史记录中订单数 | 交易函数 |
| HistorySelect | 为服务器时间的指定周期检索交易和订单历史记录 | 交易函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| HistorySelectByPosition | 请求指定仓位标识符交易的历史记录 | 交易函数 |
| iBars | 返回可用于历史记录的相应交易品种和周期的柱形图的数量 | 时间序列和指标访问 |
| iBarShift | 根据时间搜索柱形图。这个函数返回与指定时间对应的柱形图指数 | 时间序列和指标访问 |
| iClose | 返回对应的图表上柱形图的收盘价（通过'shift'参数表示） | 时间序列和指标访问 |
| iHigh | 返回对应的图表上柱形图的最高价（通过'shift'参数表示） | 时间序列和指标访问 |
| iHighest | 返回在对应图表上找到的最高值的指数（相对于当前柱形图的转移） | 时间序列和指标访问 |
| iLow | 返回对应的图表上柱形图的最低价（通过'shift'参数表示） | 时间序列和指标访问 |
| iLowest | 返回在对应图表上找到的最小值的指数（相对于当前柱形图的转移） | 时间序列和指标访问 |
| iOpen | 返回对应的图表上柱形图的开盘价（通过'shift'参数表示） | 时间序列和指标访问 |
| iTime | 返回对应的图表上柱形图的开盘时间（通过'shift'参数表示） | 时间序列和指标访问 |
| iTickVolume | 返回对应的图表上柱形图的报价量（通过'shift'参数表示） | 时间序列和指标访问 |
| iRealVolume | 返回对应的图表上柱形图的真实交易量（通过'shift'参数表示） | 时间序列和指标访问 |
| iVolume | 返回对应的图表上柱形图的报价量（通过'shift'参数表示） | 时间序列和指标访问 |
| iSpread | 返回对应的图表上柱形图的点差值（通过'shift'参数表示） | 时间序列和指标访问 |
| iAD | 累积/分配 | 技术指标 |
| iADX | 平均定向指数 | 技术指标 |
| iADXWilder | 韦尔达平均定向指数 | 技术指标 |
| iAlligator | 鳄鱼指标 | 技术指标 |
| iAMA | 适合的移动平均数 | 技术指标 |
| iAO | 动量震荡指标 | 技术指标 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| iATR | 平均真实区域 | 技术指标 |
| iBands | 布林带 | 技术指标 |
| iBearsPower | 熊市 | 技术指标 |
| iBullsPower | 牛市 | 技术指标 |
| iBWMFI | 威廉姆斯的市场便利指标 | 技术指标 |
| iCCI | 商品通道指数 | 技术指标 |
| iChaikin | 蔡金摆动指标 | 技术指标 |
| iCustom | 自定义指标 | 技术指标 |
| iDEMA | 双指数移动平均线 | 技术指标 |
| iDeMarker | 指标 | |
| iEnvelopes | 轨道线指标 | 技术指标 |
| iForce | 强力指数 | 技术指标 |
| iFractals | 分形学 | 技术指标 |
| iFrAMA | 分形学适合移动平均数 | 技术指标 |
| iGator | 鳄鱼振荡器 | 技术指标 |
| iIchimoku | 一目均衡图 | 技术指标 |
| iMA | 移动平均数 | 技术指标 |
| iMACD | 移动平均聚散指标 | 技术指标 |
| iMFI | 货币流量指标 | 技术指标 |
| iMomentum | 动量指标 | 技术指标 |
| IndicatorCreate | 返回通过MqlParam参量类型数据创建的指定技术指标处理器 | 时间序列和指标访问 |
| IndicatorParameters | 基于指定的句柄，返回指标输入参数的数量，以及参数的值和类型 | 时间序列和指标访问 |
| IndicatorRelease | 如果没有其他，移除指标处理器并释放指标计算模块 | 时间序列和指标访问 |
| IndicatorSetDouble | 设置双精度型指标属性值 | 自定义指标 |
| IndicatorSetInteger | 设置整型型指标属性值 | 自定义指标 |
| IndicatorSetString | 设置字符串指标属性值 | 自定义指标 |
| IntegerToString | 整型转换成当前长度字符串 | 函数转换 |
| iOBV | 平衡交易量 | 技术指标 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| iOsMA | 移动平均振荡指标（MACD柱状图） | 技术指标 |
| iRSI | 相对强弱指标 | 技术指标 |
| iRVI | 相对活力指标 | 技术指标 |
| iSAR | 抛物转向系统 | 技术指标 |
| IsStopped | 如果mql5程序被令停止操作，返回true | 检测 |
| iStdDev | 标准偏差 | 技术指标 |
| iStochastic | 随机摆动指标 | 技术指标 |
| iTEMA | 三倍指数移动平均数 | 技术指标 |
| iTriX | 三倍指数移动平均数振荡指标 | 技术指标 |
| iVIDyA | 动态平均数便利指标 | 技术指标 |
| iVolumes | 成交量 | 技术指标 |
| iWPR | 威廉指数 | 技术指标 |
| log | 返回自然对数 | 数学函数 |
| log10 | 以10为底返回对数 | 数学函数 |
| MarketBookAdd | 提供选定交易品种市场深度开盘，签署DOM更改的接收通告 | 市场信息 |
| MarketBookGet | 返回包括指定交易品种市场深度记录的结构数组MqlBookInfo | 市场信息 |
| MarketBookRelease | 提供选定交易品种市场深度收盘，取消DOM更改的接收通告的签署。 | 市场信息 |
| MathAbs | 返回指定数值的绝对值（绝对值） | 数学函数 |
| MathArccos | 返回x弧度的反余弦 | 数学函数 |
| MathArcsin | 返回x弧度的反正弦 | 数学函数 |
| MathArctan | 返回x弧度的反正切 | 数学函数 |
| MathCeil | 从上面返回最靠近的整数数值 | 数学函数 |
| MathCos | 返回数字余弦 | 数学函数 |
| MathExp | 返回数字指数 | 数学函数 |
| MathFloor | 从下面返回最靠近的整数数值 | 数学函数 |
| MathIsValidNumber | 检测实数真实型 | 数学函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| MathLog | 返回自然对数 | 数学函数 |
| MathLog10 | 以10为底返回对数 | 数学函数 |
| MathMax | 返回两个数值的最大值 | 数学函数 |
| MathMin | 返回两个数值的最小值 | 数学函数 |
| MathMod | 两个数值相除后返回实余数 | 数学函数 |
| MathPow | 从基数升到额定输出 | 数学函数 |
| MathRand | 0-32767范围返回随机值 | 数学函数 |
| MathRound | 四舍五入到最近整数值 | 数学函数 |
| MathSin | 返回正弦数 | 数学函数 |
| MathSqrt | 返回平方根 | 数学函数 |
| MathSrand | 设置开始点生成系列随机数 | 数学函数 |
| MathTan | 返回正切数 | 数学函数 |
| MessageBox | 创建，显示消息盒子，然后管理 | 普通函数 |
| MQLInfoInteger | 返回运行mq5程序相关属性的整数值 | 检测 |
| MQLInfoString | 返回运行mq5程序相关属性的字符串值 | 检测 |
| MT5Initialize | 建立与MetaTrader 5程序端的连接 | MetaTrader Python模块 |
| MT5Shutdown | 关于之前建立的与MetaTrader 5程序端的连接 | MetaTrader Python模块 |
| MT5TerminalInfo | 获取所连接的MetaTrader 5程序端的状态和参数 | MetaTrader Python模块 |
| MT5Version | 返回MetaTrader 5程序端版本 | MetaTrader Python模块 |
| MT5CopyRatesFrom | 从指定日期开始从MetaTrader 5程序端获取柱形图 | MetaTrader Python模块 |
| MT5CopyRatesFromPos | 从指定索引开始从MetaTrader 5程序端获取柱形图 | MetaTrader Python模块 |
| MT5CopyRatesRange | 从MetaTrader 5程序端获取指定日期范围内的柱形图 | MetaTrader Python模块 |
| MT5CopyTicksFrom | 从指定日期开始从MetaTrader 5程序端获取报价 | MetaTrader Python模块 |
| MT5CopyTicksRange | 从MetaTrader 5程序端获取指定日期范围内的报价 | MetaTrader Python模块 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| NormalizeDouble | 浮点数转换到规定的精确度 | 函数转换 |
| ObjectCreate | 创建指定图表中指定类型物件 | 物件函数 |
| ObjectDelete | 从指定图表移除指定名物件（从指定图表子窗口） | 物件函数 |
| ObjectFind | 依据名称搜索指定ID物件 | 物件函数 |
| ObjectGetDouble | 返回相应物件属性的双精度值 | 物件函数 |
| ObjectGetInteger | 返回相应物件属性的整数型值 | 物件函数 |
| ObjectGetString | 返回相应物件属性的字符串值 | 物件函数 |
| ObjectGetTimeByValue | 对指定物件价格值返回时间值 | 物件函数 |
| ObjectGetValueByTime | 对指定时间返回物件价格值 | 物件函数 |
| ObjectMove | 改变指定物件定位点坐标 | 物件函数 |
| ObjectName | 返回指定图表中相关类型物件名称（指定图表子窗口） | 物件函数 |
| ObjectsDeleteAll | 从指定图表移除指定类型所有物件（从指定图表子窗口） | 物件函数 |
| ObjectSetDouble | 设置相应物件属性值 | 物件函数 |
| ObjectSetInteger | 设置相应物件属性值 | 物件函数 |
| ObjectSetString | 设置相应物件属性值 | 物件函数 |
| ObjectsTotal | 返回指定图表中指定类型物件数（指定图表子窗口） | 物件函数 |
| OnStart | 当Start事件发生时调用这个函数，来执行脚本中设定的操作 | 事件处理 |
| OnInit | 当Init事件发生时在指标和EA中调用这个函数，来初始化启动的MQL5程序 | 事件处理 |
| OnDeinit | 当Deinit事件发生时在指标和EA中调用这个函数，来去初始化启动的MQL5程序 | 事件处理 |
| OnTick | 当NewTick事件发生时在EA中调用这个函数，来处理一个新报价 | 事件处理 |
| OnCalculate | 当Calculate事件发生时在指标中调用这个函数，来处理价格数据的变化 | 事件处理 |
| OnTimer | 当Timer周期性事件在固定时间间隔内由程序端生成期间，在 | 事件处理 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| | 指标和EA中调用这个函数 | |
| OnTrade | 当Trade事件在交易服务器上交易操作结束时生成期间，在EA中调用这个函数 | 事件处理 |
| OnTradeTransaction | 当TradeTransaction事件发生时在EA中调用这个函数，来处理一个交易请求执行结果 | 事件处理 |
| OnBookEvent | 当BookEvent事件发生时在EA中调用这个函数，来处理市场深度中的变化 | 事件处理 |
| OnChartEvent | 当ChartEvent事件发生时在指标和EA中调用这个函数，来处理由用户或者MQL5程序所做的图表更改 | 事件处理 |
| OnTester | 当Tester事件发生时在EA中调用这个函数，以便在对历史数据测试EA之后执行必要的操作 | 事件处理 |
| OnTesterInit | 当TesterInit事件发生时在EA中调用这个函数，以便在策略测试优化之前执行必要的操作 | 事件处理 |
| OnTesterDeinit | 在策略测试EA优化之后，当TesterDeinit事件发生时在EA中调用这个函数。 | 事件处理 |
| OnTesterPass | 当TesterPas事件发生时在EA中调用这个函数，以便在策略测试中处理EA优化期间到达的新数据框架 | 事件处理 |
| OrderCalcMargin | 用入金货币，计算指定订单类型所需的保证金 | 交易函数 |
| OrderCalcProfit | 以入金货币计算基于传递参量的利润 | 交易函数 |
| OrderCheck | 检测是否有足够资金执行所需交易操作. | 交易函数 |
| OrderGetDouble | 返回订单要求属性（双精度型） | 交易函数 |
| OrderGetInteger | 返回订单要求属性（日期时间或者整型） | 交易函数 |
| OrderGetString | 返回订单要求属性（字符串） | 交易函数 |
| OrderGetTicket | 返回相应订单的订单号 | 交易函数 |
| OrderSelect | 为下一步工作选择一订单 | 交易函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| OrderSend | 发送 交易需求 到服务器 | 交易函数 |
| OrderSendAsync | 不同步发送 交易请求 无需等待交易服务器的交易回应 | 交易函数 |
| OrdersTotal | 返回订单数 | 交易函数 |
| ParameterGetRange | 在策略测试优化EA期间，接收范围值的数据和输入变量的改变步骤 | 处理优化结果 |
| ParameterSetRange | 在策略测试优化EA期间，指定使用输入变量 ：值，改变步骤，初始值和最终值 | 处理优化结果 |
| Period | 返回当前图表时间表 | 检测 |
| PeriodSeconds | 在周期中返回秒钟数量 | 普通函数 |
| PlaySound | 播放第二个文件 | 普通函数 |
| PlotIndexGetInteger | 返回整数型指标行属性值 | 自定义指标 |
| PlotIndexSetDouble | 设置双精度型指标行属性值 | 自定义指标 |
| PlotIndexSetInteger | 设置整型指标行属性值 | 自定义指标 |
| PlotIndexSetString | 设置字符串指标行属性值 | 自定义指标 |
| Point | 返回报价货币当前交易品种的点大小 | 检测 |
| PositionGetDouble | 返回持仓要求属性（双精度） | 交易函数 |
| PositionGetInteger | 返回持仓要求属性（日期时间或者整型） | 交易函数 |
| PositionGetString | 返回持仓要求属性（字符串） | 交易函数 |
| PositionGetSymbol | 返回与持仓一致的交易品种 | 交易函数 |
| PositionSelect | 选择一个持仓进行下一步工作 | 交易函数 |
| PositionsTotal | 返回持仓数量 | 交易函数 |
| pow | 从基数升到额定输出 | 数学函数 |
| 打印 | 显示未经处理的消息 | 普通函数 |
| PrintFormat | 在目标文件中按照原来格式，格式刷建立的交易品种和值 | 普通函数 |
| rand | 0-32767范围返回随机值 | 数学函数 |
| ResetLastError | 建立预定义变量 _LastError的值到0 | 普通函数 |
| ResourceCreate | 基于数据设置创建图像资源 | 普通函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| ResourceFree | 删除 动态创建资源 (释放分配的内存) | 普通函数 |
| ResourceReadImage | 阅读来自通过 ResourceCreate() 函数创建的 或编译期间保存在EX5文件的图形资源的数据 | 普通函数 |
| ResourceSave | 将资源保存在指定文件 | 普通函数 |
| round | 四舍五入到最近整数值 | 数学函数 |
| SendFTP | 在"Publisher"标签窗口中设置发送文件地址 | 普通函数 |
| SendMail | 在"Mailbox"标签窗口中发送邮件到指定地址 | 普通函数 |
| SendNotification | 向在 "通知"标签指定 MetaQuotes ID的移动程序端发送推送通知。 | 普通函数 |
| SeriesInfoInteger | 返回历史资料信息 | 时间序列和指标访问 |
| SetIndexBuffer | 将指定指标缓冲区和一维双精度动态数组绑定一起 | 自定义指标 |
| ShortArrayToString | 复制数组部分到字符串 | 函数转换 |
| ShortToString | 转换符号代码 (双字节编码)到单字符串 | 函数转换 |
| SignalBaseGetDouble | 返回选定信号double类型属性的值 | 交易信号 |
| SignalBaseGetInteger | 返回选定信号integer类型属性的值 | 交易信号 |
| SignalBaseGetString | 返回选定信号string类型属性的值 | 交易信号 |
| SignalBaseSelect | 选择来自信号组的信号，在程序端可以得到，用于进一步的工作 | 交易信号 |
| SignalBaseTotal | 返回信号总额，在程序端可得 | 交易信号 |
| SignalInfoGetDouble | 返回信号复制设置的double类型属性的值 | 交易信号 |
| SignalInfoGetInteger | 返回信号复制设置的integer类型属性的值 | 交易信号 |
| SignalInfoGetString | 返回信号复制设置的string类型属性的值 | 交易信号 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| SignalInfoSetDouble | 设置信号复制设置的double类型属性的值 | 交易信号 |
| SignalInfoSetInteger | 设置信号复制设置的integer类型属性的值 | 交易信号 |
| SignalSubscribe | 订阅交易信号 | 交易信号 |
| SignalUnsubscribe | 取消订阅 | 交易信号 |
| sin | 返回正弦数 | 数学函数 |
| Sleep | 延迟执行当前EA交易或者脚本的规定间隔 | 普通函数 |
| SocketCreate | 创建具有指定标识的套接并返回其句柄 | 网络函数 |
| SocketClose | 关闭套接 | 网络函数 |
| SocketConnect | 通过超时控制连接到服务器 | 网络函数 |
| SocketIsConnected | 检查套接当前是否已连接 | 网络函数 |
| SocketIsReadable | 获得可以从套接读取的字节数 | 网络函数 |
| SocketIsWritable | 检查在当前时间是否可以将数据写入套接 | 网络函数 |
| SocketTimeouts | 为系统套接对象设置数据接收和发送超时 | 网络函数 |
| SocketRead | 从套接读取数据 | 网络函数 |
| SocketSend | 将数据写入套接 | 网络函数 |
| SocketTlsHandshake | 通过TLS Handshake协议启动与指定主机的安全TLS (SSL)连接 | 网络函数 |
| SocketTlsCertificate | 获得用于安全网络连接的证书数据 | 网络函数 |
| SocketTlsRead | 读取来自安全TLS连接的数据 | 网络函数 |
| SocketTlsReadAvailable | 读取来自安全TLS连接的所有可用数据 | 网络函数 |
| SocketTlsSend | 通过安全TLS连接发送数据 | 网络函数 |
| sqrt | 返回平方根 | 数学函数 |
| srand | 设置开始点生成系列随机数 | 数学函数 |
| StringAdd | 在给出子连接上添加字符串 | 字符串函数 |
| StringBufferLen | 为字符串返回分配式缓冲器的大小 | 字符串函数 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| StringCompare | 比较两个字符串并且如果第一个字符串大于第二个返回1；0-如果两个字符串相等；-1（负一）-如果第一个字符串小于第二个 | 字符串函数 |
| StringConcatenate | 实现一串参数传递 | 字符串函数 |
| StringFill | 通过挑选出来的交易品种填满指定字符串 | 字符串函数 |
| StringFind | 在字符串里搜索子串 | 字符串函数 |
| StringFormat | 按照现格式转换数字到字符串 | 函数转换 |
| StringGetCharacter | 在规定字符串位置返回数字值 | 字符串函数 |
| StringInit | 通过指定交易品种初始化字符串并提供指定字符串长度 | 字符串函数 |
| StringLen | 在字符串里返回交易品种数字 | 字符串函数 |
| StringReplace | 通过交易品种集合序列替代字符串发现的所有子字符串 | 字符串函数 |
| StringSetCharacter | 在指定位置返回复制后的交易品种的改变值 | 字符串函数 |
| StringSplit | 在指定字符串通过指定分隔符得到子字符串，返回所得子字符串的数量 | 字符串函数 |
| StringSubstr | 从指定位置提取文本串 | 字符串函数 |
| StringToCharArray | 符号-wise 复制双字节编码到ANSI转换的字符串，到选定的无符字符型数组 | 函数转换 |
| StringToColor | 转换"R,G,B" 字符串或者颜色名的字符串到颜色类型值 | 函数转换 |
| StringToDouble | 转换包括数字代表符的字符串到双精度数字 | 函数转换 |
| StringToInteger | 转换包括数字代表符的字符串到整型数字 | 函数转换 |
| StringToLower | 通过存储单位把所有交易品种中已选的字符串转为小写 | 字符串函数 |
| StringToShortArray | 符号-wise复制字符串到选定的无符号短整型部分数组 | 函数转换 |
| StringToTime | 转换包括 "yyyy.mm.dd [hh:mi]"格式的时间或者日期字符串到日期时间型 | 函数转换 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| StringToUpper | 通过存储单位把所有交易品种中已选的字符串转为资本 | 字符串函数 |
| StringTrimLeft | 在字符串的左边切断线路供应字符，空间和标号 | 字符串函数 |
| StringTrimRight | 在字符串的右边切断线路供应字符，空间和标号 | 字符串函数 |
| StructToTime | MqlDate Time结构类型变量转化到日期时间值 | 日期和时间 |
| Symbol | 返回当前图表的交易品种的名称 | 检测 |
| SymbolInfoDouble | 为相关属性返回交易品种双精度值 | 市场信息 |
| SymbolInfoInteger | 为相关属性返回指定交易品种的整数型值（长整型，日期时间，整型或者布尔型） | 市场信息 |
| SymbolInfoMarginRate | 根据订单类型和方向返回预付款率 | 市场信息 |
| SymbolInfoSessionQuote | 平日允许为指定交易品种接收指定报价期的起止时间 | 市场信息 |
| SymbolInfoSessionTrade | 平日允许为指定交易品种接收指定交易期的起止时间 | 市场信息 |
| SymbolInfoString | 为相关属性返回指定交易品种的字符串类型值 | 市场信息 |
| SymbolInfoTick | 返回MqlTick型变量中指定交易品种当前价格。 | 市场信息 |
| SymbolIsSynchronized | 检测程序端选择的交易品种的数据与交易服务器的数据是否同步。 | 市场信息 |
| SymbolName | 返回指定交易品种名称 | 市场信息 |
| SymbolSelect | 市场报价窗口选择或者去除交易品种 | 市场信息 |
| SymbolsTotal | 返回有效交易品种数量（选择市场报价或者全部） | 市场信息 |
| tan | 返回正切数 | 数学函数 |
| TerminalClose | 命令客户端完成操作 | 普通函数 |
| TerminalInfoDouble | 返回mql5程序环境相关属性的双精度值 | 检测 |

© 2000-2025, MetaQuotes Ltd.

| 函数 | 功能 | 章节 |
| --- | --- | --- |
| TerminalInfoInteger | 返回mql5程序环境相关属性的整数值 | 检测 |
| TerminalInfoString | 返回mql5程序环境相关属性的字符串值 | 检测 |
| TesterStatistics | 返回基于测试结果计算的指定统计的值 | 普通函数 |
| TextGetSize | 返回当前字体设置 的字符串宽度和高度 | 物件函数 |
| TextOut | 文本转到专为创建图形资源而设计的自定义数组 (缓冲区) | 物件函数 |
| TextSetFont | 设置使用绘制方法展示文本的字体 (默认使用Arial20) | 物件函数 |
| TimeCurrent | 日期时间格式返回服务器最后一个可知时间 (最后一个报价收据时间) | 日期和时间 |
| TimeDaylightSavings | 返回日节省时间切换符号 | 日期和时间 |
| TimeGMT | 通过客户端运行的计算机本地时间以日期时间格式返回GMT日节省时间。 | 日期和时间 |
| TimeGMTOffset | 以秒计算返回当前GMT和计算机本地的不同，也包括夏令时切换 | 日期和时间 |
| TimeLocal | 以日期时间格式返回本地计算机时间 | 日期和时间 |
| TimeToString | 转换01.01.1970值到"yyyy.mm.dd hh:mi"字符串格式 | 函数转换 |
| TimeToStruct | 日期时间值转换到MqlDateTime结构类型变量 | 日期和时间 |
| TimeTradeServer | 返回交易服务器当前计算时间 | 日期和时间 |
| UninitializeReason | 返回无法初始化原因代码 | 检测 |
| WebRequest | 发送 HTTP 请求到指定服务器 | 普通函数 |
| ZeroMemory | 通过引用重设变量传递，该变量可以是任何类型，除了分类和结构都有构造函数 | 普通函数 |

© 2000-2025, MetaQuotes Ltd.

# MQL5 常量列表

MQL5所有常量都是按照字母顺序排列的。

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| __DATE__ | 文件编译日期，无时间（小时，分钟，秒等于0） | 打印 |
| __DATETIME__ | 文件编译日期和时间 | 打印 |
| __FILE__ | 当前编辑文件的名称 | 打印 |
| __FUNCSIG__ | 宏所在的函数的签名。函数的完整描述记录可用于识别重载函数很 | 打印 |
| __FUNCTION__ | 函数名称，宏主 | 打印 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 体位置 | |
| __LINE__ | 源代码中的字符串数量，宏放置的位置 | 打印 |
| __MQLBUILD__, __MQL5BUILD__ | 编译器构造数量 | 打印 |
| __PATH__ | 当前正在编译的文件的绝对路径 | 打印 |
| ACCOUNT_ASSETS | 账户的流动资产 | AccountInfoDouble |
| ACCOUNT_BALANCE | 存入货币账结余 | AccountInfoDouble |
| ACCOUNT_COMMISSION_BLOCKED | 账户当前结冻的手续费金额 | AccountInfoDouble |
| ACCOUNT_COMPANY | 提供账户的公司名称 | AccountInfoString |
| ACCOUNT_CREDIT | 存入货币 | AccountInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 时账户亏空 | |
| ACCOUNT_CURRENCY | 账户货币 | AccountInfoString |
| ACCOUNT_EQUITY | 存入货币时账户权益 | AccountInfoDouble |
| ACCOUNT_LEVERAGE | 账户杠杆 | AccountInfoInteger |
| ACCOUNT_LIABILITIES | 账户的流动负债 | AccountInfoDouble |
| ACCOUNT_LIMIT_ORDERS | 活跃待办订单最大允许量 | AccountInfoInteger |
| ACCOUNT_LOGIN | 账户数量 | AccountInfoInteger |
| ACCOUNT_MARGIN | 存入货币时账户保证金使用 | AccountInfoDouble |
| ACCOUNT_MARGIN_FREE | 存入货币时账户可用保证金 | AccountInfoDouble |
| ACCOUNT_MARGIN_INITIAL | 初始预付款。账户保留的金额以涵 | AccountInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 盖所有挂单的预付款 | |
| ACCOUNT_MARGIN_LEVEL | 账户保证金水平仪百分比形式制定 | AccountInfoDouble |
| ACCOUNT_MARGIN_MAINTENANCE | 维持预付款。账户保留的最低净值以涵盖所有持仓的最低额 | AccountInfoDouble |
| ACCOUNT_MARGIN_SO_CALL | 保证金调用水平，依据建立的ACCOUNT_MARGIN_SO_MODE，以百分比形式或存入货币时期表示 | AccountInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ACCOUNT_MARGIN_SO_MODE | 建立最少使用保证金方式 | AccountInfoInteger |
| ACCOUNT_MARGIN_SO_SO | 保证金停用水平，依据建立的ACCOUNT_MARGIN_SO_MODE，以百分比形式或存入货币时期表示 | AccountInfoDouble |
| ACCOUNT_NAME | 用户名 | AccountInfoString |
| ACCOUNT_PROFIT | 存入货币账户当前利润 | AccountInfoDouble |
| ACCOUNT_SERVER | 交易服务器名称 | AccountInfoString |
| ACCOUNT_STOPOUT_MODE_MONEY | 账户以货币方式截止 | AccountInfoInteger |
| ACCOUNT_STOPOUT_MODE_PERCENT | 账户以半分比 | AccountInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 方式截止 | |
| ACCOUNT_TRADE_ALLOWED | 允许活期账户交易 | AccountInfoInteger |
| ACCOUNT_TRADE_EXPERT | 允许EA交易 | AccountInfoInteger |
| ACCOUNT_TRADE_MODE | 账户交易方式 | AccountInfoInteger |
| ACCOUNT_TRADE_MODE_CONTEST | 竞争账户 | AccountInfoInteger |
| ACCOUNT_TRADE_MODE_DEMO | 样本账户 | AccountInfoInteger |
| ACCOUNT_TRADE_MODE_REAL | 真实账户 | AccountInfoInteger |
| ALIGN_CENTER | 集中（仅用于编辑对象） | ObjectSetInteger, ObjectGetInteger, ChartScreenShot |
| ALIGN_LEFT | 左对齐 | ObjectSetInteger, ObjectGetInteger, ChartScreenShot |
| ALIGN_RIGHT | 右对齐 | ObjectSetInteger, ObjectGetInteger, ChartScreenShot |
| ANCHOR_CENTER | 对象中间严格定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_LEFT | 左侧定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_LEFT_LOWER | 左下角定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_LEFT_UPPER | 左上角定位点 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ANCHOR_LOWER | 下方定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_RIGHT | 右侧定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_RIGHT_LOWER | 右下方定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_RIGHT_UPPER | 右上角定位点 | ObjectSetInteger, ObjectGetInteger |
| ANCHOR_UPPER | 上方定位点 | ObjectSetInteger, ObjectGetInteger |
| BASE_LINE | 干线 | 指标线 |
| BOOK_TYPE_BUY | 买入命令（出价） | MqlBookInfo |
| BOOK_TYPE_BUY_MARKET | 市场的买入订单 | MqlBookInfo |
| BOOK_TYPE_SELL | 卖出命令（要价） | MqlBookInfo |
| BOOK_TYPE_SELL_MARKET | 市场的卖出订单 | MqlBookInfo |
| BORDER_FLAT | 平面造型 | ObjectSetInteger, ObjectGetInteger |
| BORDER_RAISED | 突出造型 | ObjectSetInteger, ObjectGetInteger |
| BORDER_SUNKEN | 凹陷造型 | ObjectSetInteger, ObjectGetInteger |
| CHAR_MAX | 最大值，可以 | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 代表图表类型 | |
| CHAR_MIN | 最小值，可以代表图表类型 | 数值类型常量 |
| CHART_AUTOSCROLL | 自动移向图表的右边框 | ChartSetInteger, ChartGetInteger |
| CHART_BARS | 以序列字符陈列 | ChartSetInteger |
| CHART_BEGIN | 图表开始（最原始价位） | ChartNavigate |
| CHART_BRING_TO_TOP | 显示其他图表之上的图表 | ChartSetInteger, ChartGetInteger |
| CHART_CANDLES | 以日语蜡烛台陈列 | ChartSetInteger |
| CHART_COLOR_ASK | 要价水平颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_BACKGROUND | 图表背景颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_BID | 出价水平颜色 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| CHART_COLOR_CANDLE_BEAR | 承受烛台主体颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_CANDLE_BULL | 大型烛台主体颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_CHART_DOWN | 下降字节、阴影和支撑烛台的颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_CHART_LINE | 折线图颜色和日语烛台躲闪颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_CHART_UP | 上升字节、阴影和大型烛台整体边界的颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_FOREGROUND | 轴线、缩放和OHLC线的颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_GRID | 网格颜色 | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_LAST | 最后执行交易价格水平 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 线颜色（Last） | |
| CHART_COLOR_STOP_LEVEL | 停止订购水平颜色（斩仓和获利） | ChartSetInteger, ChartGetInteger |
| CHART_COLOR_VOLUME | 成交量颜色和开仓水平 | ChartSetInteger, ChartGetInteger |
| CHART_COMMENT | 图表中的评论文本 | ChartSetString, ChartGetString |
| CHART_CURRENT_POS | 当前位置 | ChartNavigate |
| CHART_DRAG_TRADE_LEVELS | 用鼠标在图表上拖拽交易水平的权限。拖拽模式默认启用（true值） | ChartSetInteger, ChartGetInteger |
| CHART_EVENT_MOUSE_MOVE | 发送鼠标移动和鼠标点击事件的通知（CHARTEVEN | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | T_MOUSE_MOVE) 到图表上的所有 mql5 程序 | |
| CHART_EVENT_OBJECT_CREATE | 发送创建新对象的事件通知 (CHARTEVENT_OBJECT_CREATE) 到图表上的所有 mql5 程序 | ChartSetInteger, ChartGetInteger |
| CHART_EVENT_OBJECT_DELETE | 发送删除对象的事件通知 (CHARTEVENT_OBJECT_DELETE) 到图表上的所有 mql5 程序 | ChartSetInteger, ChartGetInteger |
| CHART_FIRST_VISIBLE_BAR | 图表中第一可见字 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 节字符。字节索引同于时序列。 | |
| CHART_FIXED_MAX | 固定图表最大值 | ChartSetDouble, ChartGetDouble |
| CHART_FIXED_MIN | 固定图表最小值 | ChartSetDouble, ChartGetDouble |
| CHART_FIXED_POSITION | 图表以百分比定位左边界的位置。在水平时间轴用小灰白三角标记定位的图表。只有在禁止进入订单号时图表自动滚到右侧它才会显示（见CHART_AUTO | ChartSetDouble, ChartGetDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | SCROLL property)。放大缩小时，定位的柱保留原位。 | |
| CHART_FOREGROUND | 背景的价格图表 | ChartSetInteger, ChartGetInteger |
| CHART_HEIGHT_IN_PIXELS | 图表像素高度 | ChartSetInteger, ChartGetInteger |
| CHART_IS_OBJECT | 识别“图表”(OBJ_CHART)对象 - 图形对象返回true。真实图表返回false | ChartSetInteger, ChartGetInteger |
| CHART_LINE | 收盘价格线型显示 | ChartSetInteger |
| CHART_MODE | 图表类型（蜡烛台、字节或线） | ChartSetInteger, ChartGetInteger |
| CHART_MOUSE_SCROLL | 使用鼠标左键水平 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 滚动图表。如果以下属性的值设为true也可以垂直滚动：CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR | |
| CHART_POINTS_PER_BAR | 测量相关的每个字节 | ChartSetDouble, ChartGetDouble |
| CHART_PRICE_MAX | 图表最大值 | ChartSetDouble, ChartGetDouble |
| CHART_PRICE_MIN | 图表最小值 | ChartSetDouble, ChartGetDouble |
| CHART_SCALE | 测量 | ChartSetInteger, ChartGetInteger |
| CHART_SCALE_PT_PER_BAR | 每字节指定相关测量 | ChartSetInteger, ChartGetInteger |
| CHART_SCALEFIX | 固定标盘模式 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| CHART_SCALEFIX_11 | 测量方式1:1 | ChartSetInteger, ChartGetInteger |
| CHART_SHIFT | 左边缩进价格图表 | ChartSetInteger, ChartGetInteger |
| CHART_SHIFT_SIZE | 大小在右边百分比边界从零字节缩进 | ChartSetDouble, ChartGetDouble |
| CHART_SHOW_ASK_LINE | 在图表水平线上显示要价值 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_BID_LINE | 在图表水平线上显示出价值 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_DATE_SCALE | 显示图表的时间比例 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_GRID | 在图表中显示网格 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_LAST_LINE | 在图表水平线上显示最终值 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_OBJECT_DESCR | 弹出图解 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 对象摘要 | |
| CHART_SHOW_OHLC | 在左上角显示OHLC值 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_ONE_CLICK | 显示"一键交易"图表面板 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_PERIOD_SEP | 在相邻周期显示垂直分离器 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_PRICE_SCALE | 显示图表的价格比例 | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_TRADE_LEVELS | 在图表中显示交易水平（开仓水平、斩仓、获利和代办订单） | ChartSetInteger, ChartGetInteger |
| CHART_SHOW_VOLUMES | 在图表中显示成交量 | ChartSetInteger, ChartGetInteger |
| CHART_VISIBLE_BARS | 图表上显示的 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 字节数量 | |
| CHART_VOLUME_HIDE | 交易量隐藏 | ChartSetInteger |
| CHART_VOLUME_REAL | 交易成交量 | ChartSetInteger |
| CHART_VOLUME_TICK | 最小价格成交量 | ChartSetInteger |
| CHART_WIDTH_IN_BARS | 以字节转发图表 | ChartSetInteger, ChartGetInteger |
| CHART_WIDTH_IN_PIXELS | 像素转发图表 | ChartSetInteger, ChartGetInteger |
| CHART_WINDOW_HANDLE | 处理图表窗口（HWND） | ChartSetInteger, ChartGetInteger |
| CHART_WINDOW_IS_VISIBLE | 预览窗口可见性 | ChartSetInteger, ChartGetInteger |
| CHART_WINDOW_YDISTANCE | 指标子窗口的上帧与主图表窗口的上帧之间的距离，沿着Y轴垂直，以像素为单 | ChartSetInteger, ChartGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 位。在鼠标事件情况下，根据图表主窗口的坐标传递光标坐标，而将指标子窗口的图形对象坐标设置相对于子窗口的左上角。需要该值是为了将主图表的绝对坐标转换为子窗口的本地坐标，以便图形对象的正确工作，该坐标设置位置相 | |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 对于子窗口帧的左上角。 | |
| CHART_WINDOWS_TOTAL | 图表窗口总数，包括指标预览窗口 | ChartSetInteger, ChartGetInteger |
| CHARTEVENT_CHART_CHANGE | 通过属性对话框改变图表大小或更改图表属性 | OnChartEvent |
| CHARTEVENT_CLICK | 点击一个图表 | OnChartEvent |
| CHARTEVENT_CUSTOM | 事件的首写字符在自定义事件中 | OnChartEvent |
| CHARTEVENT_CUSTOM_LAST | 事件的最后一个字符在自定义事件中 | OnChartEvent |
| CHARTEVENT_KEYDOWN | 打键次数 | OnChartEvent |
| CHARTEVENT_MOUSE_MOVE | 鼠标移动， | OnChartEvent |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 鼠标点击（如果CHART_EVENT_MOUSE_MOVE=true 为图表设置） | |
| CHARTEVENT_OBJECT_CHANGE | 图解对象通过属性对话改变性质 | OnChartEvent |
| CHARTEVENT_OBJECT_CLICK | 点击图解对象 | OnChartEvent |
| CHARTEVENT_OBJECT_CREATE | 图解对象创建（如果CHART_EVENT_OBJECT_CREATE=true 为图表设置） | OnChartEvent |
| CHARTEVENT_OBJECT_DELETE | 图解对象删除（如果CHART_EVENT_OBJECT_DELETE=true 为图 | OnChartEvent |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 表设置） | |
| CHARTEVENT_OBJECT_DRAG | 拖放图解对象 | OnChartEvent |
| CHARTEVENT_OBJECT_ENDEDIT | 在图表对象编辑里进行文本末尾编辑 | OnChartEvent |
| CHARTS_MAX | 在终端与开仓图表类似的最大可能值 | 其他常量 |
| CHIKOUSPAN_LINE | 延迟线 | 指标线 |
| clrAliceBlue | 爱丽丝蓝 | 网页颜色 |
| clrAntiqueWhite | 古董白 | 网页颜色 |
| clrAqua | 浅绿色 | 网页颜色 |
| clrAquamarine | 宝石蓝 | 网页颜色 |
| clrBeige | 米色 | 网页颜色 |
| clrBisque | 淡褐色 | 网页颜色 |
| clrBlack | 黑色 | 网页颜色 |
| clrBlanchedAlmond | 杏仁白 | 网页颜色 |
| clrBlue | 蓝色 | 网页颜色 |
| clrBlueViolet | 紫罗兰色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrBrown | 棕色 | 网页颜色 |
| clrBurlyWood | 原木色 | 网页颜色 |
| clrCadetBlue | 灰蓝色 | 网页颜色 |
| clrChartreuse | 黄绿色 | 网页颜色 |
| clrChocolate | 巧克力色 | 网页颜色 |
| clrCoral | 珊瑚色 | 网页颜色 |
| clrCornflowerBlue | 矢车菊蓝 | 网页颜色 |
| clrCornsilk | 玉米丝色 | 网页颜色 |
| clrCrimson | 深红色 | 网页颜色 |
| clrDarkBlue | 深蓝色 | 网页颜色 |
| clrDarkGoldenrod | 暗金黄色 | 网页颜色 |
| clrDarkGray | 深灰色 | 网页颜色 |
| clrDarkGreen | 深绿色 | 网页颜色 |
| clrDarkKhaki | 深卡其色 | 网页颜色 |
| clrDarkOliveGreen | 深橄榄绿色 | 网页颜色 |
| clrDarkOrange | 暗橘色 | 网页颜色 |
| clrDarkOrchid | 深兰花紫 | 网页颜色 |
| clrDarkSalmon | 深橙红色 | 网页颜色 |
| clrDarkSeaGreen | 暗海蓝色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrDarkSlateBlue | 暗灰蓝色 | 网页颜色 |
| clrDarkSlateGray | 深青灰色 | 网页颜色 |
| clrDarkTurquoise | 暗宝石绿 | 网页颜色 |
| clrDarkViolet | 暗紫色 | 网页颜色 |
| clrDeepPink | 深粉色 | 网页颜色 |
| clrDeepSkyBlue | 深天蓝色 | 网页颜色 |
| clrDimGray | 暗灰色 | 网页颜色 |
| clrDodgerBlue | 宝蓝色 | 网页颜色 |
| clrFireBrick | 火砖色 | 网页颜色 |
| clrForestGreen | 深林绿 | 网页颜色 |
| clrGainsboro | 亮灰色 | 网页颜色 |
| clrGold | 金色 | 网页颜色 |
| clrGoldenrod | 金麒麟色 | 网页颜色 |
| clrGray | 灰色 | 网页颜色 |
| clrGreen | 绿色 | 网页颜色 |
| clrGreenYellow | 黄绿色 | 网页颜色 |
| clrHoneydew | 浅粉红 | 网页颜色 |
| clrHotPink | 艳粉色 | 网页颜色 |
| clrIndianRed | 印度红 | 网页颜色 |
| clrIndigo | 靛蓝色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrIvory | 象牙白 | 网页颜色 |
| clrKhaki | 卡其色 | 网页颜色 |
| clrLavender | 薰衣草色 | 网页颜色 |
| clrLavenderBlush | 淡紫红 | 网页颜色 |
| clrLawnGreen | 草绿色 | 网页颜色 |
| clrLemonChiffon | 柠檬绸色 | 网页颜色 |
| clrLightBlue | 淡蓝色 | 网页颜色 |
| clrLightCoral | 浅珊瑚色 | 网页颜色 |
| clrLightCyan | 浅青色 | 网页颜色 |
| clrLightGoldenrod | 浅金黄色 | 网页颜色 |
| clrLightGray | 浅灰色 | 网页颜色 |
| clrLightGreen | 淡绿色 | 网页颜色 |
| clrLightPink | 淡粉红色 | 网页颜色 |
| clrLightSalmon | 浅肉色 | 网页颜色 |
| clrLightSeaGreen | 浅海蓝色 | 网页颜色 |
| clrLightSkyBlue | 浅天蓝色 | 网页颜色 |
| clrLightSlateGray | 浅青灰色 | 网页颜色 |
| clrLightSteelBlue | 浅钢蓝色 | 网页颜色 |
| clrLightYellow | 淡黄色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrLime | 石灰色 | 网页颜色 |
| clrLimeGreen | 石灰绿色 | 网页颜色 |
| clrLinen | 亚麻色 | 网页颜色 |
| clrMagenta | 洋红色 | 网页颜色 |
| clrMaroon | 栗色 | 网页颜色 |
| clrMediumAquamarine | 蓝绿色 | 网页颜色 |
| clrMediumBlue | 中蓝色 | 网页颜色 |
| clrMediumOrchid | 中粉紫色 | 网页颜色 |
| clrMediumPurple | 中紫色 | 网页颜色 |
| clrMediumSeaGreen | 中海蓝色 | 网页颜色 |
| clrMediumSlateBlue | 中暗蓝色 | 网页颜色 |
| clrMediumSpringGreen | 中春绿色 | 网页颜色 |
| clrMediumTurquoise | 中绿宝石色 | 网页颜色 |
| clrMediumVioletRed | 中紫罗兰色 | 网页颜色 |
| clrMidnightBlue | 午夜蓝 | 网页颜色 |
| clrMintCream | 薄荷乳色 | 网页颜色 |
| clrMistyRose | 粉玫瑰红 | 网页颜色 |
| clrMoccasin | 鹿皮色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrNavajoWhite | 纳瓦白 | 网页颜色 |
| clrNavy | 海军蓝 | 网页颜色 |
| clrNONE | 颜色缺乏 | 其他常量 |
| clrOldLace | 老花色 | 网页颜色 |
| clrOlive | 橄榄色 | 网页颜色 |
| clrOliveDrab | 草绿色 | 网页颜色 |
| clrOrange | 橙色 | 网页颜色 |
| clrOrangeRed | 橙红色 | 网页颜色 |
| clrOrchid | 兰花紫色 | 网页颜色 |
| clrPaleGoldenrod | 淡金黄色 | 网页颜色 |
| clrPaleGreen | 淡绿色 | 网页颜色 |
| clrPaleTurquoise | 淡粉蓝色 | 网页颜色 |
| clrPaleVioletRed | 苍紫罗蓝色 | 网页颜色 |
| clrPapayaWhip | Papaya Whip | 网页颜色 |
| clrPeachPuff | 桃红色 | 网页颜色 |
| clrPeru | 秘鲁色 | 网页颜色 |
| clrPink | 粉色 | 网页颜色 |
| clrPlum | 紫红色 | 网页颜色 |
| clrPowderBlue | 粉末蓝 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrPurple | 紫色 | 网页颜色 |
| clrRed | 红色 | 网页颜色 |
| clrRosyBrown | 玫瑰粽 | 网页颜色 |
| clrRoyalBlue | 皇家蓝 | 网页颜色 |
| clrSaddleBrown | 重褐色 | 网页颜色 |
| clrSalmon | 鲑肉色 | 网页颜色 |
| clrSandyBrown | 沙褐色 | 网页颜色 |
| clrSeaGreen | 海绿色 | 网页颜色 |
| clrSeashell | 海贝色 | 网页颜色 |
| clrSienna | 赭色 | 网页颜色 |
| clrSilver | 银色 | 网页颜色 |
| clrSkyBlue | 天蓝色 | 网页颜色 |
| clrSlateBlue | 石蓝色 | 网页颜色 |
| clrSlateGray | 青灰色 | 网页颜色 |
| clrSnow | 白血色 | 网页颜色 |
| clrSpringGreen | 春天绿n | 网页颜色 |
| clrSteelBlue | 钢青色 | 网页颜色 |
| clrTan | 茶色 | 网页颜色 |
| clrTeal | 青色 | 网页颜色 |
| clrThistle | 苍紫色 | 网页颜色 |
| clrTomato | 西红柿色 | 网页颜色 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| clrTurquoise | 绿松石e | 网页颜色 |
| clrViolet | 紫罗兰色 | 网页颜色 |
| clrWheat | 小麦色 | 网页颜色 |
| clrWhite | 白色 | 网页颜色 |
| clrWhiteSmoke | 白烟 | 网页颜色 |
| clrYellow | 黄色 | 网页颜色 |
| clrYellowGreen | 黄绿色 | 网页颜色 |
| CORNER_LEFT_LOWER | 中央左边在图表左下角 | ObjectSetInteger, ObjectGetInteger |
| CORNER_LEFT_UPPER | 中央坐标在图表左上角 | ObjectSetInteger, ObjectGetInteger |
| CORNER_RIGHT_LOWER | 中央坐标在图表右下角 | ObjectSetInteger, ObjectGetInteger |
| CORNER_RIGHT_UPPER | 中央左边在图表右上角 | ObjectSetInteger, ObjectGetInteger |
| CP_ACP | 当前窗口ANSI代码页 | CharArrayToString, StringToCharArray, FileOpen |
| CP_MACCP | 当前系统Macintosh | CharArrayToString, StringToCharArray, FileOpen |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 代码页。**注释：**值主要在早期创建的程序代码中使用，现在没有用处了，新的Macintosh电脑使用统一码。 | |
| CP_OEMCP | 当前系统OEM代码页 | CharArrayToString, StringToCharArray, FileOpen |
| CP_SYMBOL | 交易品种代码页 | CharArrayToString, StringToCharArray, FileOpen |
| CP_THREAD_ACP | 当前线路使用ANSI窗口代码 | CharArrayToString, StringToCharArray, FileOpen |
| CP_UTF7 | UTF-7代码页 | CharArrayToString, StringToCharArray, FileOpen |
| CP_UTF8 | UTF-8代码页 | CharArrayToString, StringToCharArray, FileOpen |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| CRYPT_AES128 | AES 128位密钥加密（16字节） | CryptEncode, CryptDecode |
| CRYPT_AES256 | AES 256位密钥加密（32字节） | CryptEncode, CryptDecode |
| CRYPT_ARCH_ZIP | ZIP 归档 | CryptEncode, CryptDecode |
| CRYPT_BASE64 | BASE64 | CryptEncode, CryptDecode |
| CRYPT_DES | DES 56位密钥加密（7字节） | CryptEncode, CryptDecode |
| CRYPT_HASH_MD5 | MD5 HASH计算 | CryptEncode, CryptDecode |
| CRYPT_HASH_SHA1 | SHA1 HASH caculation | CryptEncode, CryptDecode |
| CRYPT_HASH_SHA256 | SHA256 HASH caculation | CryptEncode, CryptDecode |
| DBL_DIG | 双精度型有效小数位数字 | 数值类型常量 |
| DBL_EPSILON | 最小值，满足条件 | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 1.0+DBL_EPSILON != 1.0（双精度类型） | |
| DBL_MANT_DIG | 双精度型尾数算在内的二进制 | 数值类型常量 |
| DBL_MAX | 最大值，可以代表双精度型 | 数值类型常量 |
| DBL_MAX_10_EXP | 双精度型指数角度最大小数值 | 数值类型常量 |
| DBL_MAX_EXP | 双精度型指数角度最大二进制值 | 数值类型常量 |
| DBL_MIN | 最小正值，可以代表双精度型 | 数值类型常量 |
| DBL_MIN_10_EXP | 双精度型指数角度 | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 最小小数 | |
| DBL_MIN_EXP | 双精度型指数角度最小二进制值 | 数值类型常量 |
| DEAL_COMMENT | 订单佣金 | HistoryDealGetString |
| DEAL_COMMISSION | 订单佣金关 | HistoryDealGetDouble |
| DEAL_ENTRY | 交易条目—允许进入、允许输出、颠倒 | HistoryDealGetInteger |
| DEAL_ENTRY_IN | 进入 | HistoryDealGetInteger |
| DEAL_ENTRY_INOUT | 反面 | HistoryDealGetInteger |
| DEAL_ENTRY_OUT | 出局 | HistoryDealGetInteger |
| DEAL_MAGIC | 订单幻数（参见ORDER_MAGIC） | HistoryDealGetInteger |
| DEAL_ORDER | 交易订单号 | HistoryDealGetInteger |
| DEAL_POSITION_ID | 仓位标识符在开仓、修正或者交易发生改变 | HistoryDealGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 时，每个位置都有一个独特的标识符，为交易对象它被委派到所有执行订单，在全部方位使用期 | |
| DEAL_PRICE | 成交价格 | HistoryDealGetDouble |
| DEAL_PROFIT | 交易利润 | HistoryDealGetDouble |
| DEAL_SWAP | 关闭的积累交换 | HistoryDealGetDouble |
| DEAL_SYMBOL | 订单交易品种 | HistoryDealGetString |
| DEAL_TIME | 交易时间 | HistoryDealGetInteger |
| DEAL_TIME_MSC | 以毫秒为单位计算01.01.1970以来交易执行的时间 | HistoryDealGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| DEAL_TYPE | 交易类型 | HistoryDealGetInteger |
| DEAL_TYPE_BALANCE | 结算 | HistoryDealGetInteger |
| DEAL_TYPE_BONUS | 奖金 | HistoryDealGetInteger |
| DEAL_TYPE_BUY | 买入 | HistoryDealGetInteger |
| DEAL_TYPE_BUY_CANCELED | 取消的买入交易。可能会有之前执行的买入交易被取消的情况。这样的话，之前执行的交易类型（DEAL_TYPE_BUY）会被改为DEAL_TYPE_BUY_CANCELED，并且其利润/亏损归零。执行获得的利润/亏损会使用单独的结余操作被收费/取款。 | HistoryDealGetInteger |
| DEAL_TYPE_CHARGE | 额外金额 | HistoryDealGetInteger |
| DEAL_TYPE_COMMISSION | 额外手续费 | HistoryDealGetInteger |
| DEAL_TYPE_COMMISSION_AGENT_DAILY | 日代理手续费 | HistoryDealGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| DEAL_TYPE_COMMISSION_AGENT_MONTHLY | 月代理手续费 | HistoryDealGetInteger |
| DEAL_TYPE_COMMISSION_DAILY | 日手续费 | HistoryDealGetInteger |
| DEAL_TYPE_COMMISSION_MONTHLY | 月手续费 | HistoryDealGetInteger |
| DEAL_TYPE_CORRECTION | 修正 | HistoryDealGetInteger |
| DEAL_TYPE_CREDIT | 信贷 | HistoryDealGetInteger |
| DEAL_TYPE_INTEREST | 利率 | HistoryDealGetInteger |
| DEAL_TYPE_SELL | 卖出 | HistoryDealGetInteger |
| DEAL_TYPE_SELL_CANCELED | 取消的卖出交易。可能会有之前执行的卖出交易被取消的情况。这样的话，之前执行的交易类型 (DEAL_TYPE_SELL) 会被改为 DEAL_TYPE_SELL_CANCELED，并且其利润/亏损归零。执行获得的利润/亏损会使用单独的结余操作被收费/取款。 | HistoryDealGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| DEAL_VOLUME | 成交量 | HistoryDealGetDouble |
| DRAW_ARROW | 画箭头 | 绘画风格 |
| DRAW_BARS | 以字符序列展示 | 绘画风格 |
| DRAW_CANDLES | 以蜡烛台序列展示 | 绘画风格 |
| DRAW_COLOR_ARROW | 描绘多色箭头 | 绘画风格 |
| DRAW_COLOR_BARS | 多色字节 | 绘画风格 |
| DRAW_COLOR_CANDLES | 多色蜡烛台 | 绘画风格 |
| DRAW_COLOR_HISTOGRAM | 从0线开始的多色直方图 | 绘画风格 |
| DRAW_COLOR_HISTOGRAM2 | 两个指标缓冲区的多色直方图 | 绘画风格 |
| DRAW_COLOR_LINE | 多色线 | 绘画风格 |
| DRAW_COLOR_SECTION | 多色分段 | 绘画风格 |
| DRAW_COLOR_ZIGZAG | 多色之字形 | 绘画风格 |
| DRAW_FILLING | 两层间的 | 绘画风格 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 颜色 | |
| DRAW_HISTOGRAM | 从0位线画柱状图 | 绘画风格 |
| DRAW_HISTOGRAM2 | 两个指标命令的柱状图 | 绘画风格 |
| DRAW_LINE | 画线 | 绘画风格 |
| DRAW_NONE | 没有画线 | 绘画风格 |
| DRAW_SECTION | 线条 | 绘画风格 |
| DRAW_ZIGZAG | 之字设计允许垂直界面 | 绘画风格 |
| ELLIOTT_CYCLE | 循环 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_GRAND_SUPERCYCLE | 庞大的超周期 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_INTERMEDIATE | 媒介 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_MINOR | 次要的 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_MINUETTE | 波浪 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_MINUTE | 分钟 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_PRIMARY | 原色 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_SUBMINUETTE | 潜波浪 | ObjectSetInteger, ObjectGetInteger |
| ELLIOTT_SUPERCYCLE | 超周期 | ObjectSetInteger, ObjectGetInteger |
| EMPTY_VALUE | 指标缓冲区的空值 | 其他常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_ACCOUNT_WRONG_PROPERTY | 错误的账户属性ID | GetLastError |
| ERR_ARRAY_BAD_SIZE | 要求数组大小超过2GB | GetLastError |
| ERR_ARRAY_RESIZE_ERROR | 数组没有足够空间重置，或者没有能够改变静态数组的大小 | GetLastError |
| ERR_BOOKS_CANNOT_ADD | 不能添加市场深度 | GetLastError |
| ERR_BOOKS_CANNOT_DELETE | 不能移动市场深度 | GetLastError |
| ERR_BOOKS_CANNOT_GET | 深度市场数据未获得 | GetLastError |
| ERR_BOOKS_CANNOT_SUBSCRIBE | 从市场深度接收新数据时出错 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_BUFFERS_NO_MEMORY | 没有足够内存建立指标缓冲区 | GetLastError |
| ERR_BUFFERS_WRONG_INDEX | 错误指标缓冲区索引 | GetLastError |
| ERR_CANNOT_CLEAN_DIRECTORY | 清除目录失败（或许一个或多个文件阻塞而作操失败） | GetLastError |
| ERR_CANNOT_DELETE_DIRECTORY | 目录不能更改 | GetLastError |
| ERR_CANNOT_DELETE_FILE | 文件删除错误 | GetLastError |
| ERR_CANNOT_OPEN_FILE | 文件打开错误 | GetLastError |
| ERR_CHAR_ARRAY_ONLY | 必须是图表型数据 | GetLastError |
| ERR_CHART_CANNOT_CHANGE | 改变图表交易品种或周 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 期失败 | |
| ERR_CHART_CANNOT_CREATE_TIMER | 增加定时器失败 | GetLastError |
| ERR_CHART_CANNOT_OPEN | 图表打开错误 | GetLastError |
| ERR_CHART_INDICATOR_CANNOT_ADD | 错误添加指标到图表 | GetLastError |
| ERR_CHART_INDICATOR_CANNOT_DEL | 删除图表指标的错误 | GetLastError |
| ERR_CHART_INDICATOR_NOT_FOUND | 指定图表上未发现指标 | GetLastError |
| ERR_CHART_NAVIGATE_FAILED | 错误操作图表 | GetLastError |
| ERR_CHART_NO_EXPERT | 图表中没有EA交易可以处理事件 | GetLastError |
| ERR_CHART_NO_REPLY | 图表不能回应 | GetLastError |
| ERR_CHART_NOT_FOUND | 图表未发现 | GetLastError |
| ERR_CHART_SCREENSHOT_FAILED | 错误生成 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 截屏 | |
| ERR_CHART_TEMPLATE_FAILED | 错误申请模板 | GetLastError |
| ERR_CHART_WINDOW_NOT_FOUND | 未找到窗口包含的指标 | GetLastError |
| ERR_CHART_WRONG_ID | 错误图表ID | GetLastError |
| ERR_CHART_WRONG_PARAMETER | 图表使用函数的错误参数值 | GetLastError |
| ERR_CHART_WRONG_PROPERTY | 错误图表属性ID | GetLastError |
| ERR_CUSTOM_WRONG_PROPERTY | 自定义指标的错误ID | GetLastError |
| ERR_DIRECTORY_NOT_EXIST | 目录不存在 | GetLastError |
| ERR_DOUBLE_ARRAY_ONLY | 必须是双精度类型数组 | GetLastError |
| ERR_FILE_BINSTRINGSIZE | 字符串大小必须制定，因为文件以二 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 进制打开 | |
| ERR_FILE_CACHEBUFFER_ERROR | 没有足够内存隐藏阅读 | GetLastError |
| ERR_FILE_CANNOT_REWRITE | 文件不能重写 | GetLastError |
| ERR_FILE_IS_DIRECTORY | 没有文件，只有目录 | GetLastError |
| ERR_FILE_ISNOT_DIRECTORY | 只有文件，没有目录 | GetLastError |
| ERR_FILE_NOT_EXIST | 文件不存在 | GetLastError |
| ERR_FILE_NOTBIN | 文件必须以二进制打开 | GetLastError |
| ERR_FILE_NOTCSV | 文件必须以CSV格式打开 | GetLastError |
| ERR_FILE_NOTTOREAD | 文件必须打开阅读 | GetLastError |
| ERR_FILE_NOTTOWRITE | 文件必须打开编辑 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_FILE_NOTTXT | 文件必须以文本形式打开 | GetLastError |
| ERR_FILE_NOTTXTORCSV | 文件必须以文本或者CSV打开 | GetLastError |
| ERR_FILE_READERROR | 文件阅读错误 | GetLastError |
| ERR_FILE_WRITEERROR | 将resource写入文件失败 | GetLastError |
| ERR_FLOAT_ARRAY_ONLY | 必须是浮点型数组 | GetLastError |
| ERR_FTP_SEND_FAILED | 通过ftp发送文件失败 | GetLastError |
| ERR_FUNCTION_NOT_ALLOWED | 系统函数不允许调用 | GetLastError |
| ERR_GLOBALVARIABLE_EXISTS | 相同名称客户全局变量已经存在 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_GLOBALVARIABLE_NOT_FOUND | 客户端全局变量未找到 | GetLastError |
| ERR_HISTORY_NOT_FOUND | 需求历史未找到 | GetLastError |
| ERR_HISTORY_WRONG_PROPERTY | 历史属性的错误ID | GetLastError |
| ERR_INCOMPATIBLE_ARRAYS | 复制不兼容数组，字符串数组只能复制成字符串数组，和数字数组-只在数字数组中 | GetLastError |
| ERR_INCOMPATIBLE_FILE | 文本文件必须以字符串数组打开，或其他数组-二进制 | GetLastError |
| ERR_INDICATOR_CANNOT_ADD | 错误申请指标 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 到图表 | |
| ERR_INDICATOR_CANNOT_APPLY | 指标不能适应另一指标 | GetLastError |
| ERR_INDICATOR_CANNOT_CREATE | 不能创建指标 | GetLastError |
| ERR_INDICATOR_CUSTOM_NAME | 在数组中第一参量的一与定义指标自定相同 | GetLastError |
| ERR_INDICATOR_DATA_NOT_FOUND | 需求数据未找到 | GetLastError |
| ERR_INDICATOR_NO_MEMORY | 没有足够内存添加指标 | GetLastError |
| ERR_INDICATOR_PARAMETER_TYPE | 当建立指标时，数组中无效参量类型 | GetLastError |
| ERR_INDICATOR_PARAMETERS_MISSING | 当建立指标时没有参量 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_INDICATOR_UNKNOWN_SYMBOL | 未知交易品种 | GetLastError |
| ERR_INDICATOR_WRONG_HANDLE | 错误指标处理 | GetLastError |
| ERR_INDICATOR_WRONG_INDEX | 错误索引需求的指标缓冲区 | GetLastError |
| ERR_INDICATOR_WRONG_PARAMETERS | 当建立指标时使用的错误参量数 | GetLastError |
| ERR_INT_ARRAY_ONLY | 必须是整型数组 | GetLastError |
| ERR_INTERNAL_ERROR | 意外内部错误 | GetLastError |
| ERR_INVALID_ARRAY | 数组错误类型，错误大小，或者动态数组的损害对象 | GetLastError |
| ERR_INVALID_DATETIME | 无效日期和/时间 | GetLastError |
| ERR_INVALID_FILEHANDLE | 文件处理 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 关闭，或者根本没打开 | |
| ERR_INVALID_PARAMETER | 当调用系统函数时的错误参量 | GetLastError |
| ERR_INVALID_POINTER | 错误指针 | GetLastError |
| ERR_INVALID_POINTER_TYPE | 错误指针类型 | GetLastError |
| ERR_LONG_ARRAY_ONLY | 必须是长型数组 | GetLastError |
| ERR_MAIL_SEND_FAILED | 发送邮件失败 | GetLastError |
| ERR_MARKET_LASTTIME_UNKNOWN | 时间的最后标志未识别（无标志） | GetLastError |
| ERR_MARKET_NOT_SELECTED | 市场报价中未被挑选出来的交易品种 | GetLastError |
| ERR_MARKET_SELECT_ERROR | 在市场报价中 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 添加或删除交易品种的错误 | |
| ERR_MARKET_UNKNOWN_SYMBOL | 未知交易品种 | GetLastError |
| ERR_MARKET_WRONG_PROPERTY | 交易品种属性的错误标识符 | GetLastError |
| ERR_MQL5_WRONG_PROPERTY | 程序属性错误标识符 | GetLastError |
| ERR_NO_STRING_DATE | 字符串无日期 | GetLastError |
| ERR_NOT_ENOUGH_MEMORY | 没有足够空间执行系统函数 | GetLastError |
| ERR_NOTIFICATION_SEND_FAILED | 发送通知失败 | GetLastError |
| ERR_NOTIFICATION_TOO_FREQUENT | 发送通知过于频繁 | GetLastError |
| ERR_NOTIFICATION_WRONG_PARAMETER | 发送通知的无效参数 - 空字符串或 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | NULL被传递到SendNotification()函数 | |
| ERR_NOTIFICATION_WRONG_SETTINGS | 错误的程序端通知设置（未指定ID或未设置权限） | GetLastError |
| ERR_NOTINITIALIZED_STRING | 没有初始化字符串 | GetLastError |
| ERR_NUMBER_ARRAYS_ONLY | 一定是数字数组 | GetLastError |
| ERR_OBJECT_ERROR | 图解对象的误差 | GetLastError |
| ERR_OBJECT_GETDATE_FAILED | 不能与值相一致获得日期 | GetLastError |
| ERR_OBJECT_GETVALUE_FAILED | 不能与日期相一致获得值 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_OBJECT_NOT_FOUND | 未发现图解对象 | GetLastError |
| ERR_OBJECT_WRONG_PROPERTY | 图解对象属性的错误ID | GetLastError |
| ERR_ONEDIM_ARRAYS_ONLY | 一定是一维数组 | GetLastError |
| ERR_OPENCL_BUFFER_CREATE | 创建OpenCL缓存失败 | GetLastError |
| ERR_OPENCL_CONTEXT_CREATE | 创建OpenCL语境的错误 | GetLastError |
| ERR_OPENCL_EXECUTE | OpenCL程序运行时间错误 | GetLastError |
| ERR_OPENCL_INTERNAL | 运行OpenCL时发生的内部错误 | GetLastError |
| ERR_OPENCL_INVALID_HANDLE | 无效的OpenCL句柄 | GetLastError |
| ERR_OPENCL_KERNEL_CREATE | 创建OpenCL | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | kernel的错误 | |
| ERR_OPENCL_NOT_SUPPORTED | 该计算机不支持OpenCL函数 | GetLastError |
| ERR_OPENCL_PROGRAM_CREATE | 编译OpenCL程序时发生的错误 | GetLastError |
| ERR_OPENCL_QUEUE_CREATE | 在OpenCL创建运行队列失败 | GetLastError |
| ERR_OPENCL_SET_KERNEL_PARAMETER | 当为OpenCL kernel设置参数时发生的错误 | GetLastError |
| ERR_OPENCL_TOO_LONG_KERNEL_NAME | 过长的内核名称(OpenCL kernel) | GetLastError |
| ERR_OPENCL_WRONG_BUFFER_OFFSET | 无效的OpenCL缓 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 存补偿 | |
| ERR_OPENCL_WRONG_BUFFER_SIZE | 无效的OpenCL缓存大小 | GetLastError |
| ERR_PLAY_SOUND_FAILED | 声音播放失败 | GetLastError |
| ERR_RESOURCE_NAME_DUPLICATED | 动态和静态resource匹配的名称 | GetLastError |
| ERR_RESOURCE_NAME_IS_TOO_LONG | resource名称超过63个字符 | GetLastError |
| ERR_RESOURCE_NOT_FOUND | 该名称的resource在EX5没有找到 | GetLastError |
| ERR_RESOURCE_UNSUPPORTED_TYPE | 不支持的resource类型或其大小超过16Mb | GetLastError |
| ERR_SERIES_ARRAY | 不能使用时序列 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_SHORT_ARRAY_ONLY | 必须是短型数组 | GetLastError |
| ERR_SMALL_ARRAY | 太小数组，启动位置在数组外 | GetLastError |
| ERR_SMALL_ASSERIES_ARRAY | 以AS_SERIES形式接收数据，不够大 | GetLastError |
| ERR_STRING_OUT_OF_MEMORY | 没有足够空间建立字符串 | GetLastError |
| ERR_STRING_RESIZE_ERROR | 没有足够内存重置字符串 | GetLastError |
| ERR_STRING_SMALL_LEN | 字符串太短 | GetLastError |
| ERR_STRING_TIME_ERROR | 错误转变字符串到日期 | GetLastError |
| ERR_STRING_TOO_BIGNUMBER | 太多数组，比 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | ULONG_MAX多 | |
| ERR_STRING_UNKNOWNTYPE | 当转换字符串时的未知数据类型 | GetLastError |
| ERR_STRING_ZEROADDED | 字符串末尾添加0，无用操作 | GetLastError |
| ERR_STRINGPOS_OUTOFRANGE | 字符串外部位置 | GetLastError |
| ERR_STRUCT_WITHOBJECTS_ORCLASS | 结构包括字符串和/或动态数组和/或结构对象和/或分类 | GetLastError |
| ERR_SUCCESS | 操作成功完成 | GetLastError |
| ERR_TERMINAL_WRONG_PROPERTY | 客户端属性错误标识符 | GetLastError |
| ERR_TOO_LONG_FILENAME | 太长文件名 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_TOO_MANY_FILES | 不能同时打开超过64个文件 | GetLastError |
| ERR_TOO_MANY_FORMATTERS | 格式说明符数量比参量多 | GetLastError |
| ERR_TOO_MANY_PARAMETERS | 参量数多于格式说明符 | GetLastError |
| ERR_TRADE_DEAL_NOT_FOUND | 未找到订单 | GetLastError |
| ERR_TRADE_DISABLED | EA交易禁止交易 | GetLastError |
| ERR_TRADE_ORDER_NOT_FOUND | 未找到命令 | GetLastError |
| ERR_TRADE_POSITION_NOT_FOUND | 未找到位置 | GetLastError |
| ERR_TRADE_SEND_FAILED | 交易需求发送失败 | GetLastError |
| ERR_TRADE_WRONG_PROPERTY | 错误交易属性ID | GetLastError |
| ERR_USER_ERROR_FIRST | User defined 错误以该代 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 码起始 | |
| ERR_WEBREQUEST_CONNECT_FAILED | 连接指定URL失败 | GetLastError |
| ERR_WEBREQUEST_INVALID_ADDRESS | 无效URL | GetLastError |
| ERR_WEBREQUEST_REQUEST_FAILED | HTTP请求失败 | GetLastError |
| ERR_WEBREQUEST_TIMEOUT | 超时 | GetLastError |
| ERR_WRONG_DIRECTORYNAME | 错误目录名称 | GetLastError |
| ERR_WRONG_FILEHANDLE | 错误文件处理 | GetLastError |
| ERR_WRONG_FILENAME | 无效文件名 | GetLastError |
| ERR_WRONG_FORMATSTRING | 无效格式字符串 | GetLastError |
| ERR_WRONG_INTERNAL_PARAMETER | 客户端函数内部调用的错误参数 | GetLastError |
| ERR_WRONG_STRING_DATE | 字符串错误日期 | GetLastError |
| ERR_WRONG_STRING_OBJECT | 破坏字符串对象 | GetLastError |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ERR_WRONG_STRING_PARAMETER | 字符串类型破坏参量 | GetLastError |
| ERR_WRONG_STRING_TIME | 字符串错误时间 | GetLastError |
| ERR_ZEROSIZE_ARRAY | 0长度数组 | GetLastError |
| FILE_ACCESS_DATE | 最后访问文件的日期 | FileGetInteger |
| FILE_ANSI | ANSI类型字符串（一字节交易品种），使用FileOpen()标签 | FileOpen |
| FILE_BIN | 二进制阅读/书写模式（不需要转换字符串），使用FileOpen()标签 | FileOpen |
| FILE_COMMON | 所有客户端在共同文件 | FileOpen, FileCopy, FileMove, FileIsExist |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 夹中的文件路径 \Terminal\Common\Files，使用 FileIsExist() 函数中的 FileOpen()，FileCopy()，FileMove() 标签。 | |
| FILE_CREATE_DATE | 创建日期 | FileGetInteger |
| FILE_CSV | CSV文件（所有元素转换到恰当的字符串类型，统一编码或者ansi，并被分离器分离）使用 FileOpen() 标签 | FileOpen |
| FILE_END | 获取文件结束标志 | FileGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| FILE_EXISTS | 检查是否存在 | FileGetInteger |
| FILE_IS_ANSI | 文件打开为ANSI文件(请见FILE_ANSI) | FileGetInteger |
| FILE_IS_BINARY | 文件打开为二进制文件(请见FILE_BIN) | FileGetInteger |
| FILE_IS_COMMON | 在所有程序端共享的文件夹中打开文件(请见FILE_COMMON) | FileGetInteger |
| FILE_IS_CSV | 文件打开为CSV文件(请见FILE_CSV) | FileGetInteger |
| FILE_IS_READABLE | 打开文件为可读文件(请见 | FileGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | FILE_READ) | |
| FILE_IS_TEXT | 文件打开为文本文件(请见FILE_TXT) | FileGetInteger |
| FILE_IS_WRITABLE | 打开文件为可写入文件(请见FILE_WRITE) | FileGetInteger |
| FILE_LINE_END | 获取线型结束标志 | FileGetInteger |
| FILE_MODIFY_DATE | 最近的更改日期 | FileGetInteger |
| FILE_POSITION | 文件指针的位置 | FileGetInteger |
| FILE_READ | 文件只读，使用FileOpen()标签。当打开文件规格需要使用FILE_WRITE | FileOpen |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 和/或FILE_READ | |
| FILE_REWRITE | 使用FileCopy()和FileMove()函数可以重新输入文件，文件应该在书写时存在或者被打开，否则文件不能打开。 | FileCopy, FileMove |
| FILE_SHARE_READ | 多个程序共享阅读，使用FileOpen()标签，但是当打开文件时，它不能替代表示FILE_WRITE和/或FILE_READ | FileOpen |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 标识的必要性。 | |
| FILE_SHARE_WRITE | 多个程序分享输入，使用FileOpen()标签，但是当打开文件时，它不能替代表示FILE_WRITE和/或FILE_READ标识的必要性。 | FileOpen |
| FILE_SIZE | 文件的字节大小 | FileGetInteger |
| FILE_TXT | 简单文本文件（与CSV文件相同，但不考虑分离器），使用FileOpen()标签 | FileOpen |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| FILE_UNICODE | UNICODE类型字符串（两字节交易品种），使用FileOpen()标签 | FileOpen |
| FILE_WRITE | 文件书写开放，使用FileOpen()标签。当打开文件规格需要使用FILE_WRITE和/或FILE_READ | FileOpen |
| FLT_DIG | 浮点型有效角度数位 | 数值类型常量 |
| FLT_EPSILON | 最小值，满足因素：1.0+DBL_EPSILON != 1.0（浮点类型） | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| FLT_MANT_DIG | 浮点型二进制计算点数 | 数值类型常量 |
| FLT_MAX | 最大值，能代表浮点类型 | 数值类型常量 |
| FLT_MAX_10_EXP | 浮点型指数角度最大数值 | 数值类型常量 |
| FLT_MAX_EXP | 浮点型最大二进制值 | 数值类型常量 |
| FLT_MIN | 最小正值，能代表浮点类型 | 数值类型常量 |
| FLT_MIN_10_EXP | 浮点型指数角度最小数值 | 数值类型常量 |
| FLT_MIN_EXP | 浮点型指数角度最小二进制值 | 数值类型常量 |
| FRIDAY | 星期五 | SymbolInfoInteger, SymbolInfoSessionQuote, |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | | SymbolInfoSessionTrade |
| GANN_DOWN_TREND | 与下降趋势一致的线 | ObjectSetInteger, ObjectGetInteger |
| GANN_UP_TREND | 与上升趋势一致的线 | ObjectSetInteger, ObjectGetInteger |
| GATORJAW_LINE | 爪线 | 指标线 |
| GATORLIPS_LINE | 唇线 | 指标线 |
| GATORTEETH_LINE | 牙线 | 指标线 |
| IDABORT | 按“停止”键 | MessageBox |
| IDCANCEL | 按“取消”键 | MessageBox |
| IDCONTINUE | 按“继续”键 | MessageBox |
| IDIGNORE | 按“忽略”键 | MessageBox |
| IDNO | 按“否”键 | MessageBox |
| IDOK | 按“确认”键 | MessageBox |
| IDRETRY | 按“重试”键 | MessageBox |
| IDTRYAGAIN | 按“再试一次”键 | MessageBox |
| IDYES | 按“是”键 | MessageBox |
| IND_AC | 加速振荡器 | IndicatorCreate, IndicatorParameters |
| IND_AD | 聚集/分散 | IndicatorCreate, IndicatorParameters |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| IND_ADX | 均定向指标 | IndicatorCreate, IndicatorParameters |
| IND_ADXW | 亚当理论的ADX | IndicatorCreate, IndicatorParameters |
| IND_ALLIGATOR | 鳄鱼指标 | IndicatorCreate, IndicatorParameters |
| IND_AMA | 相应的移动平均数 | IndicatorCreate, IndicatorParameters |
| IND_AO | 动量振荡指标-AO指标 | IndicatorCreate, IndicatorParameters |
| IND_ATR | 真实波动幅度均值 | IndicatorCreate, IndicatorParameters |
| IND_BANDS | 布林线指标（AO指标） | IndicatorCreate, IndicatorParameters |
| IND_BEARS | 熊市 | IndicatorCreate, IndicatorParameters |
| IND_BULLS | 牛市 | IndicatorCreate, IndicatorParameters |
| IND_BWMFI | 市场便利指标 | IndicatorCreate, IndicatorParameters |
| IND_CCI | 顺势指标 | IndicatorCreate, IndicatorParameters |
| IND_CHAIKIN | 佳庆指标 | IndicatorCreate, IndicatorParameters |
| IND_CUSTOM | 自定义指标 | IndicatorCreate, IndicatorParameters |
| IND_DEMA | 双精度移动平均线 | IndicatorCreate, IndicatorParameters |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| IND_DEMARKER | DEM指标 | IndicatorCreate, IndicatorParameters |
| IND_ENVELOPES | 轨道线指标 | IndicatorCreate, IndicatorParameters |
| IND_FORCE | 强力指数 | IndicatorCreate, IndicatorParameters |
| IND_FRACTALS | 拼图 | IndicatorCreate, IndicatorParameters |
| IND_FRAMA | 自适拼图移动平均数 | IndicatorCreate, IndicatorParameters |
| IND_GATOR | 振荡器 | IndicatorCreate, IndicatorParameters |
| IND_ICHIMOKU | 一目均衡图 | IndicatorCreate, IndicatorParameters |
| IND_MA | 平均移动 | IndicatorCreate, IndicatorParameters |
| IND_MACD | MACD | IndicatorCreate, IndicatorParameters |
| IND_MFI | 货币流量索引 | IndicatorCreate, IndicatorParameters |
| IND_MOMENTUM | 动量 | IndicatorCreate, IndicatorParameters |
| IND_OBV | 平衡成交量 | IndicatorCreate, IndicatorParameters |
| IND_OSMA | OsMA | IndicatorCreate, IndicatorParameters |
| IND_RSI | 相对强势索引 | IndicatorCreate, IndicatorParameters |
| IND_RVI | 相对活力索引 | IndicatorCreate, IndicatorParameters |
| IND_SAR | 抛物线SAR | IndicatorCreate, IndicatorParameters |
| IND_STDDEV | 标准偏差 | IndicatorCreate, IndicatorParameters |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| IND_STOCHASTIC | 随机振荡器 | IndicatorCreate, IndicatorParameters |
| IND_TEMA | 三倍指数移动平均值 | IndicatorCreate, IndicatorParameters |
| IND_TRIX | 三倍指数移动平均值振荡器 | IndicatorCreate, IndicatorParameters |
| IND_VIDYA | 索引变量平均值 | IndicatorCreate, IndicatorParameters |
| IND_VOLUMES | 成交量 | IndicatorCreate, IndicatorParameters |
| IND_WPR | 威廉姆斯百分比幅度 | IndicatorCreate, IndicatorParameters |
| INDICATOR_CALCULATIONS | 媒介运算的辅助缓冲区 | SetIndexBuffer |
| INDICATOR_COLOR_INDEX | 颜色 | SetIndexBuffer |
| INDICATOR_DATA | 描绘数据 | SetIndexBuffer |
| INDICATOR_DIGITS | 描绘指标值的精确度 | IndicatorSetInteger |
| INDICATOR_HEIGHT | 修正指标窗口的高 | IndicatorSetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 度(预处理命令#property indicator_height) | |
| INDICATOR_LEVELCOLOR | 水平线颜色 | IndicatorSetInteger |
| INDICATOR_LEVELS | 指标窗口中的水平数量 | IndicatorSetInteger |
| INDICATOR_LEVELSTYLE | 水平线类型 | IndicatorSetInteger |
| INDICATOR_LEVELTEXT | 水平描述 | IndicatorSetString |
| INDICATOR_LEVELVALUE | 水平值 | IndicatorSetDouble |
| INDICATOR_LEVELWIDTH | 水平线厚度 | IndicatorSetInteger |
| INDICATOR_MAXIMUM | 指标窗口最大化 | IndicatorSetDouble |
| INDICATOR_MINIMUM | 指标窗口最小化 | IndicatorSetDouble |
| INDICATOR_SHORTNAME | 短指标名称 | IndicatorSetString |
| INT_MAX | 最大值，可以代表整型 | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| INT_MIN | 最小值，可以代表整型 | 数值类型常量 |
| INVALID_HANDLE | 错误处理 | 其他常量 |
| IS_DEBUG_MODE | MQL5程序操作的调试方式标志 | 其他常量 |
| IS_PROFILE_MODE | mq5-program 运作分析模式的标识 | 其他常量 |
| KIJUNSEN_LINE | 基准线 | 指标线 |
| LICENSE_DEMO | 市场付费产品的试用版仅在策略测试中工作 | MQLInfoInteger |
| LICENSE_FREE | 免费无限使用版 | MQLInfoInteger |
| LICENSE_FULL | 购买的授权版允许至少5次激活。激 | MQLInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 活次数由卖家设定。卖家可以提高允许的激活次数 | |
| LICENSE_TIME | 有期限限制的授权版 | MQLInfoInteger |
| LONG_MAX | 最大值，可以代表长型 | 数值类型常量 |
| LONG_MIN | 最小值，可以代表长型 | 数值类型常量 |
| LOWER_BAND | 最低限制 | 指标线 |
| LOWER_HISTOGRAM | 底部直方图 | 指标线 |
| LOWER_LINE | 下边线 | 指标线 |
| M_1_PI | 1/pi | 数学常量 |
| M_2_PI | 2/pi | 数学常量 |
| M_2_SQRTPI | 2/sqrt(pi) | 数学常量 |
| M_E | e | 数学常量 |
| M_LN10 | ln(10) | 数学常量 |
| M_LN2 | ln(2) | 数学常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| M_LOG10E | log10(e) | 数学常量 |
| M_LOG2E | log2(e) | 数学常量 |
| M_PI | pi | 数学常量 |
| M_PI_2 | pi/2 | 数学常量 |
| M_PI_4 | pi/4 | 数学常量 |
| M_SQRT1_2 | 1/sqrt(2) | 数学常量 |
| M_SQRT2 | sqrt(2) | 数学常量 |
| MAIN_LINE | 干线 | 指标线 |
| MB_ABORTRETRYIGNORE | 通知窗口包括三个按钮：停止，重试和忽略 | MessageBox |
| MB_CANCELTRYCONTINUE | 通知窗口包括三个按钮：取消，再试一次，继续 | MessageBox |
| MB_DEFBUTTON1 | 如果另外按钮MB_DEFBUTTON2，MB_DEFBUTTON3，或者 | MessageBox |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | MB_DEFBUTTON4 不执行，第一按钮 MB_DEFBUTTON1 —默认。 | |
| MB_DEFBUTTON2 | 默认第二按钮 | MessageBox |
| MB_DEFBUTTON3 | 默认第三按钮 | MessageBox |
| MB_DEFBUTTON4 | 默认第四按钮 | MessageBox |
| MB_ICONEXCLAMATION, MB_ICONWARNING | 惊叹/警示标志 icon | MessageBox |
| MB_ICONINFORMATION, MB_ICONASTERISK | i包围标志 | MessageBox |
| MB_ICONQUESTION | icon 疑问标志 icon | MessageBox |
| MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND | STOP 停止标志 icon | MessageBox |
| MB_OK | 通知窗口只包括一个按钮：确定默认 | MessageBox |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| MB_OKCANCEL | 通知窗口包括两个按钮：确定和取消 | MessageBox |
| MB_RETRYCANCEL | 通知窗口包括两个按钮：重试和取消 | MessageBox |
| MB_YESNO | 通知窗口包括两个按钮：是和否 | MessageBox |
| MB_YESNOCANCEL | 通知窗口包括三个按钮：是，否和取消 | MessageBox |
| MINUSDI_LINE | 线 -DI | 指标线 |
| MODE_EMA | 指数平均数 | 平滑方式 |
| MODE_LWMA | 线性平均数 | 平滑方式 |
| MODE_SMA | 简单平均 | 平滑方式 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 数 | |
| MODE_SMMA | 平滑平均数 | 平滑方式 |
| MONDAY | 星期一 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| MQL_DEBUG | 标志表示调试方式 | MQLInfoInteger |
| MQL_DLLS_ALLOWED | 允许为已生效的程序使用DLL | MQLInfoInteger |
| MQL_FRAME_MODE | 标识，表示以收集优化结果框架的模式操作的EA交易 | MQLInfoInteger |
| MQL_LICENSE_TYPE | EX5模块的许可证类型 许可证涉及EX5模块，从这里使用MQLInfoInteger(MQL_LICENSE | MQLInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | _TYPE) 发出请求。 | |
| MQL_MEMORY_LIMIT | MQL5程序最大可能的动态内存数，MB计算 | MQLInfoInteger |
| MQL_MEMORY_USED | MQL5程序使用的内存大小，MB计算 | MQLInfoInteger |
| MQL_OPTIMIZATION | 标志表示最佳化过程 | MQLInfoInteger |
| MQL_PROFILER | 标识，表示以代码分析模式操作的程序 | MQLInfoInteger |
| MQL_PROGRAM_NAME | mql5已执行程序名称 | MQLInfoString |
| MQL_PROGRAM_PATH | 以执行系统路径 | MQLInfoString |
| MQL_PROGRAM_TYPE | MQL5程序 | MQLInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 类型 | |
| MQL_SIGNALS_ALLOWED | 更改信号的权限 针对已给的执行程序 | MQLInfoInteger |
| MQL_TESTER | 标志表示测试过程 | MQLInfoInteger |
| MQL_TRADE_ALLOWED | 允许为已生效的程序交易 | MQLInfoInteger |
| MQL_VISUAL_MODE | 标志表示视觉测试过程 | MQLInfoInteger |
| NULL | 任意类型零值 | 其他常量 |
| OBJ_ALL_PERIODS | 所有时间表对象显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_ARROW | 箭头 | 物件类型 |
| OBJ_ARROW_BUY | 买入标志 | 物件类型 |
| OBJ_ARROW_CHECK | 检测标志 | 物件类型 |
| OBJ_ARROW_DOWN | 向下箭头 | 物件类型 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJ_ARROW_LEFT_PRICE | 左侧定价标签 | 物件类型 |
| OBJ_ARROW_RIGHT_PRICE | 右侧定价标签 | 物件类型 |
| OBJ_ARROW_SELL | 卖出标志 | 物件类型 |
| OBJ_ARROW_STOP | 停止标志 | 物件类型 |
| OBJ_ARROW_THUMB_DOWN | 大拇指向下 | 物件类型 |
| OBJ_ARROW_THUMB_UP | 大拇指向上 | 物件类型 |
| OBJ_ARROW_UP | 向上箭头 | 物件类型 |
| OBJ_ARROWED_LINE | 箭头线路 | 物件类型 |
| OBJ_BITMAP | 位图 | 物件类型 |
| OBJ_BITMAP_LABEL | 位图标签 | 物件类型 |
| OBJ_BUTTON | 按钮 | 物件类型 |
| OBJ_CHANNEL | 等距离通道 | 物件类型 |
| OBJ_CHART | 图表 | |
| OBJ_CYCLES | 周期线 | 物件类型 |
| OBJ_EDIT | 编辑 | 物件类型 |
| OBJ_ELLIOTWAVE3 | 埃利奥特修正波动 | 物件类型 |
| OBJ_ELLIOTWAVE5 | 埃利奥特动机波动 | 物件类型 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJ_ELLIPSE | 椭圆形 | 物件类型 |
| OBJ_EVENT | “事件”对象在经济日历对应一个事件 | 物件类型 |
| OBJ_EXPANSION | 斐波纳契扩展 | 物件类型 |
| OBJ_FIBO | 斐波纳契撤回 | 物件类型 |
| OBJ_FIBOARC | 斐波纳契弧线 | 物件类型 |
| OBJ_FIBOCHANNEL | 斐波纳契通道 | 物件类型 |
| OBJ_FIBOFAN | 斐波纳契扇形线 | 物件类型 |
| OBJ_FIBOTIMES | 斐波纳契时间周期线 | 物件类型 |
| OBJ_GANNFAN | 江恩扇形线 | 物件类型 |
| OBJ_GANNGRID | 江恩网格线 | 物件类型 |
| OBJ_GANNLINE | 江恩线 | 物件类型 |
| OBJ_HLINE | 水平线 | 物件类型 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJ_LABEL | 标签 | 物件类型 |
| OBJ_NO_PERIODS | 时间表中不引用对象 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_D1 | 以天为单位的图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H1 | 1小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H12 | 12小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H2 | 2小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H3 | 3小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H4 | 4小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H6 | 6小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_H8 | 8小时图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M1 | 1分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M10 | 10分钟图表显示 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJ_PERIOD_M12 | 12分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M15 | 15分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M2 | 2分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M20 | 20分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M3 | 3分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M30 | 30分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M4 | 4分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M5 | 5分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_M6 | 6分钟图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_MN1 | 以月为单位的图表显示 | ObjectSetInteger, ObjectGetInteger |
| OBJ_PERIOD_W1 | 以周为单位的图表显示 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJ_PITCHFORK | 安德鲁分叉线 | 物件类型 |
| OBJ_RECTANGLE | 巨型 | 物件类型 |
| OBJ_RECTANGLE_LABEL | 用于创建和设计自定义图形界面的“矩形标签”对象。 | 物件类型 |
| OBJ_REGRESSION | 线形回归通道 | 物件类型 |
| OBJ_STDDEVCHANNEL | 标准偏离通道 | 物件类型 |
| OBJ_TEXT | 文本 | 物件类型 |
| OBJ_TREND | 趋势线 | 物件类型 |
| OBJ_TRENDBYANGLE | 趋势角度 | 物件类型 |
| OBJ_TRIANGLE | 三角形 | 物件类型 |
| OBJ_VLINE | 垂直线 | 物件类型 |
| OBJPROP_ALIGN | “编辑”对象中的水平文本对齐(OBJ_EDIT) | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_ANCHOR | 图解对象 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 的定位点 | |
| OBJPROP_ANGLE | 角度。对于程序创建的，无指定角度的对象来说，它的值等同于EMPTY_VALUE | ObjectSetDouble, ObjectGetDouble |
| OBJPROP_ARROWCODE | 箭头对象代码 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_BACK | 显示背景上的对象 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_BGCOLOR | 编辑对象背景颜色OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_BMPFILE | 文件的位图标签名称另见Resources | ObjectSetString, ObjectGetString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJPROP_BORDER_COLOR | OBJ_EDIT 和 OBJ_BUTTON 对象的边界色 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_BORDER_TYPE | "矩形标签"对象的边界类型 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_CHART_ID | "图表"对象的 ID(OBJ_CHART)。它允许使用该对象的属性，好像使用图表操作中描述的函数的普通图表，但是有一些例外。 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_CHART_SCALE | 图解对象测量 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_COLOR | 颜色 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_CORNER | 图表连接到图解对象 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJPROP_CREATETIME | 创建对象时间 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_DATE_SCALE | 显示图解对象时间比例 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_DEGREE | 埃利奥特波动标记水平 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_DEVIATION | 标准偏差通道误差 | ObjectSetDouble, ObjectGetDouble |
| OBJPROP_DIRECTION | 江恩物件趋势 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_DRAWLINES | 埃利奥特波动标记排成一行 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_ELLIPSE | 显示裴波那契弧形对象的全部椭圆形 (OBJ_FIBOARC) | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_FILL | 颜色填充对象（为OBJ_RECTANGLE, | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION) | |
| OBJPROP_FONT | 字体 | ObjectSetString, ObjectGetString |
| OBJPROP_FONTSIZE | 字体大小 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_HIDDEN | 禁止在程序端菜单“图表”–“对象”–“对象列表”的对象列表中显示图形对象的名称。真实值允许隐藏列表的对象。默认下，true的设置 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 用于显示日程表事件的对象，交易历史和MQL5程序创建的对象。若要看到这种图形对象和访问他们的属性，请点击“对象列表”窗口中的“全部”按钮。 | |
| OBJPROP_LEVELCOLOR | 线路水平颜色 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_LEVELS | 水平数量 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_LEVELSTYLE | 线路水平风格 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_LEVELTEXT | 水平描述 | ObjectSetString, ObjectGetString |
| OBJPROP_LEVELVALUE | 水平值 | ObjectSetDouble, ObjectGetDouble |
| OBJPROP_LEVELWIDTH | 线路水平厚度 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJPROP_NAME | 物件名称 | ObjectSetString, ObjectGetString |
| OBJPROP_PERIOD | 图解对象时间表 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_PRICE | 价格坐标 | ObjectSetDouble, ObjectGetDouble |
| OBJPROP_PRICE_SCALE | 显示图解对象价格比例 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_RAY | 垂直线通过图表的全部窗口 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_RAY_LEFT | 向左发出射线 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_RAY_RIGHT | 向右发出射线 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_READONLY | 在编辑对象上编辑文本 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_SCALE | 比例（江恩对象和斐波纳契弧线） | ObjectSetDouble, ObjectGetDouble |
| OBJPROP_SELECTABLE | 对象可用性 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_SELECTED | 所选对象 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| OBJPROP_STATE | 按钮状态（按压/未按压） | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_STYLE | 类型 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_SYMBOL | 图表对象的交易品种 | ObjectSetString, ObjectGetString |
| OBJPROP_TEXT | 物件描述（物件中包含的文本） | ObjectSetString, ObjectGetString |
| OBJPROP_TIME | 时间坐标 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_TIMEFRAMES | 时间表上的对象可见性 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_TOOLTIP | 提示信息文本。如果没有设置属性，那么将显示程序端自动生成的提示信息。提示信息可以通过设置"\n" | ObjectSetString, ObjectGetString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | (line feed)值来禁用 | |
| OBJPROP_TYPE | 对象类型 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_WIDTH | 路线厚度 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_XDISTANCE | X轴定位点像素距离(see note) | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_XOFFSET | 图形对象“位图标签”和“位图”(OBJ_BITMAP_LABEL和OBJ_BITMAP)中的矩形可见区域左上角的X坐标。该值由原始图片左上角对应的像素设置。 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_XSIZE | 对象宽度沿着X | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 轴，以像素表示。指定用于OBJ_LABEL（只读），OBJ_BUTTON，OBJ_CHART，OBJ_BITMAP，OBJ_BITMAP_LABEL，OBJ_EDIT，OBJ_RECTANGLE_LABEL对象 | |
| OBJPROP_YDISTANCE | Y轴定位点像素距离 (see note) | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_YOFFSET | 图形对象“位图标签”和“位图”(OBJ_BITMAP_LABEL和OBJ_BITMAP)中的矩形 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 可见区域左上角的Y坐标。该值由原始图片左上角对应的像素设置。 | |
| OBJPROP_YSIZE | 对象高度沿着Y轴，以像素表示。指定用于OBJ_LABEL（只读），OBJ_BUTTON，OBJ_CHART，OBJ_BITMAP，OBJ_BITMAP_LABEL，OBJ_EDIT，OBJ_RECTANGLE_LABEL对象 | ObjectSetInteger, ObjectGetInteger |
| OBJPROP_ZORDER | 图形对象优先接收 | ObjectSetInteger, ObjectGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 点击图表事件(CHARTEVENT_CLICK)。创建对象时默认设置零值；若需要可以提高优先级。当一个接一个地应用对象时，只有级别最高的那个可以收到CHARTEVENT_CLICK事件。 | |
| ORDER_COMMENT | 评价订单 | OrderGetString, HistoryOrderGetString |
| ORDER_FILLING_FOK | 执行政策意味着订单只可以在指定额度执行。如果当 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 前市场不提供金融工具需要的额度，订单将无法执行。需要的交易量可以使用市场此刻几种可用的提供来执行。 | |
| ORDER_FILLING_IOC | 该模式意味着交易者同意在订单指定范围内，以市场可用的最大交易量执行交易。如果无法执行全部订单交易量，那么剩 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 下的交易量将被取消。 | |
| ORDER_FILLING_RETURN | 该政策只用于市场订单 (ORDER_TYPE_BUY 和 ORDER_TYPE_SELL)，限价和止损限价订单 (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT 和 ORDER_TYPE_SELL_STOP_LIMIT) 并且只用于市场或交易 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 执行的交易品种。如果部分执行市场或剩下交易量的限价订单没有取消，则是会进一步处理。为了激活ORDER_TYPE_BUY_STOP_LIMIT和ORDER_TYPE_SELL_STOP_LIMIT订单，将会创建ORDER_FILLING_RETURN执行类型相应的限价订单ORDER_TYPE_BUY_LIMIT/ | |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | ORDER_TYPE_SELL_LIMIT。 | |
| ORDER_MAGIC | EA交易的ID安置在订单之中（为确保每个EA交易安置都有独特的代码） | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_POSITION_ID | 一旦执行，位置指标马上建立一个订单。每个执行订单结果都使订单公开或修改已经存在的位置，明确指出指标的位置同时执 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 行订单 | |
| ORDER_PRICE_CURRENT | 交易品种订单的当前价格 | OrderGetDouble, HistoryOrderGetDouble |
| ORDER_PRICE_OPEN | 订单中的规定价格 | OrderGetDouble, HistoryOrderGetDouble |
| ORDER_PRICE_STOPLIMIT | 限制停止订单的限价订单的制格 | OrderGetDouble, HistoryOrderGetDouble |
| ORDER_SL | 斩仓值 | OrderGetDouble, HistoryOrderGetDouble |
| ORDER_STATE | 订单陈述 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_CANCELED | 客户撤销订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_EXPIRED | 过期订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_FILLED | 履行全部订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_PARTIAL | 履行部分订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_PLACED | 接收订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_REJECTED | 驳回订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_REQUEST_ADD | 注册订单到交 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 易系统） | |
| ORDER_STATE_REQUEST_CANCEL | 删除订单（从交易系统中删除） | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_REQUEST_MODIFY | 更改订单（改变其参数） | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_STATE_STARTED | 选中订单，并未被经纪人接收 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_SYMBOL | 交易品种订单 | OrderGetString, HistoryOrderGetString |
| ORDER_TIME_DAY | 前交易日订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_DONE | 订单执行或取消时间 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_DONE_MSC | 以毫秒为单位计算01.01.1970以来订单执行/取消的时间 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_EXPIRATION | 订单终结 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 时间 | |
| ORDER_TIME_GTC | 取消订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_SETUP | 订单设置时间 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_SETUP_MSC | 以毫秒为单位计算01.01.1970以来下订单执行的时间 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_SPECIFIED | 过期订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TIME_SPECIFIED_DAY | 订单将生效直至指定日期的23:59:59。如果该时间超出了交易期，那么该订单会在最近的交易时间内到期。 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TP | 盈利值 | OrderGetDouble, HistoryOrderGetDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| ORDER_TYPE | 订单类型 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_BUY | 市场购买订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_BUY_LIMIT | 限制买入待办订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_BUY_STOP | 停止买入待办订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_BUY_STOP_LIMIT | 在到达订单价格之上，是限制买入订单安置在停止限制价格中 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_FILLING | 订单填满类型 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_SELL | 市场卖出订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_SELL_LIMIT | 限制卖出待办订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_SELL_STOP | 停止卖出待办订单 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_TYPE_SELL_STOP_LIMIT | 在到达订单价 | OrderGetInteger, HistoryOrderGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 格之上，是限制卖出订单安置在停止限制价格中 | |
| ORDER_TYPE_TIME | 订单使用期 | OrderGetInteger, HistoryOrderGetInteger |
| ORDER_VOLUME_CURRENT | 订单当前交易量 | OrderGetDouble, HistoryOrderGetDouble |
| ORDER_VOLUME_INITIAL | 订单最初交易量 | OrderGetDouble, HistoryOrderGetDouble |
| PERIOD_CURRENT | 当前时间表 | 图表时间表 |
| PERIOD_D1 | 1 天 | 图表时间表 |
| PERIOD_H1 | 1小时 | 图表时间表 |
| PERIOD_H12 | 12 小时 | 图表时间表 |
| PERIOD_H2 | 2 小时 | 图表时间表 |
| PERIOD_H3 | 3 小时 | 图表时间表 |
| PERIOD_H4 | 4 小时 | 图表时间表 |
| PERIOD_H6 | 6小时 | 图表时间表 |
| PERIOD_H8 | 8小时 | 图表时间表 |
| PERIOD_M1 | 1 分钟 | 图表时间表 |
| PERIOD_M10 | 10 分钟 | 图表时间表 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| PERIOD_M12 | 12 分钟 | 图表时间表 |
| PERIOD_M15 | 15 分钟 | 图表时间表 |
| PERIOD_M2 | 2 分钟 | 图表时间表 |
| PERIOD_M20 | 20 分钟 | 图表时间表 |
| PERIOD_M3 | 3 分钟 | 图表时间表 |
| PERIOD_M30 | 30 分钟 | 图表时间表 |
| PERIOD_M4 | 4 分钟 | 图表时间表 |
| PERIOD_M5 | 5 分钟 | 图表时间表 |
| PERIOD_M6 | 6 分钟 | 图表时间表 |
| PERIOD_MN1 | 1月 | 图表时间表 |
| PERIOD_W1 | 1 周 | 图表时间表 |
| PLOT_ARROW | DRAW_ARROW 类型箭头代码 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_ARROW_SHIFT | DRAW_ARROW 类型垂直箭头转换 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_COLOR_INDEXES | 颜色数量 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_DRAW_BEGIN | 在 DataWindow 中没有描绘和值的原始 | PlotIndexSetInteger, PlotIndexGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 字节数量 | |
| PLOT_DRAW_TYPE | 图解建筑类型 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_EMPTY_VALUE | 测绘空值，没有绘图 | PlotIndexSetDouble |
| PLOT_LABEL | 显示在DataWindow中的指标图解系列的名称. 当使用复杂的需要多个指标缓冲区来显示的图形风格时，可以使用","作为分隔符来指定每个缓冲区的名称。示例代码显示在DRAW | PlotIndexSetString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | _CANDLES | |
| PLOT_LINE_COLOR | 包含绘画颜色的缓冲区指标 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_LINE_STYLE | 画线类型 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_LINE_WIDTH | 画线层次 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_SHIFT | 在字节中与时间轴一起描绘的指标转换 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLOT_SHOW_DATA | 在DataWindow中显示建筑值标志 | PlotIndexSetInteger, PlotIndexGetInteger |
| PLUSDI_LINE | 线+DI | 指标线 |
| POINTER_AUTOMATIC | 任意类型对象指针自动创建（不使用new()） | CheckPointer |
| POINTER_DYNAMIC | new()操作创建的对 | CheckPointer |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 象指针 | |
| POINTER_INVALID | 错误指针 | CheckPointer |
| POSITION_COMMENT | 方位注释 | PositionGetString |
| POSITION_COMMISSION | 佣金 | PositionGetDouble |
| POSITION_IDENTIFIER | 位置标识符是独一的数字，对于每个新的开仓位来说是制定的且在整个使用期内是不可更改的，翻转位置不能改变它的标识符。 | PositionGetInteger |
| POSITION_MAGIC | 位置幻数（参看ORDER_MAGIC） | PositionGetInteger |
| POSITION_PRICE_CURRENT | 开仓当前价位 | PositionGetDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| POSITION_PRICE_OPEN | 方位开盘价格 | PositionGetDouble |
| POSITION_PROFIT | 当前利润 | PositionGetDouble |
| POSITION_SL | 开仓止损水平 | PositionGetDouble |
| POSITION_SWAP | 积累交换 | PositionGetDouble |
| POSITION_SYMBOL | 交易品种位置 | PositionGetString |
| POSITION_TIME | 开盘时间位置 | PositionGetInteger |
| POSITION_TIME_MSC | 以毫秒为单位计算01.01.1970以来持仓的时间 | PositionGetInteger |
| POSITION_TIME_UPDATE | 以秒为单位计算01.01.1970以来持仓更改的时间 | PositionGetInteger |
| POSITION_TIME_UPDATE_MSC | 以毫秒为单位计算01.01.1970以来 | PositionGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 持仓更改的时间 | |
| POSITION_TP | 开仓获利水平 | PositionGetDouble |
| POSITION_TYPE | 位置类型 | PositionGetInteger |
| POSITION_TYPE_BUY | 买入 | PositionGetInteger |
| POSITION_TYPE_SELL | 卖出 | PositionGetInteger |
| POSITION_VOLUME | 方位成交量 | PositionGetDouble |
| PRICE_CLOSE | 收盘价格 | 价格常数 |
| PRICE_HIGH | 一个时期的最高价格 | 价格常数 |
| PRICE_LOW | 一个时期的最低价格 | 价格常数 |
| PRICE_MEDIAN | 中间值 (高+低)/2 | 价格常数 |
| PRICE_OPEN | 开盘价格 | 价格常数 |
| PRICE_TYPICAL | 典型价格 (高+低+收盘价)/3 | 价格常数 |
| PRICE_WEIGHTED | 平均价格 (高 | 价格常数 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | +低 +收盘价格 +开盘价格)/4 | |
| PROGRAM_EXPERT | 专家 | MQLInfoInteger |
| PROGRAM_INDICATOR | 指标 | |
| PROGRAM_SCRIPT | 脚本 | MQLInfoInteger |
| REASON_ACCOUNT | 由于账户设置的改变，会产生激活另一个账户或者重新连接交易服务器的情况。 | UninitializeReason, OnDeinit |
| REASON_CHARTCHANGE | 更改交易品种或图表周期 | UninitializeReason, OnDeinit |
| REASON_CHARTCLOSE | 关闭图表 | UninitializeReason, OnDeinit |
| REASON_CLOSE | 程序端已关闭 | UninitializeReason, OnDeinit |
| REASON_INITFAILED | 值代表 OnInit()处理器， | UninitializeReason, OnDeinit |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 返回非零值 | |
| REASON_PARAMETERS | 使用者改变输入参数 | UninitializeReason, OnDeinit |
| REASON_PROGRAM | 通过调用ExpertRemove()函数EA交易终止操作 | UninitializeReason, OnDeinit |
| REASON_RECOMPILE | 程序重新编译 | UninitializeReason, OnDeinit |
| REASON_REMOVE | 从图表中删除程序 | UninitializeReason, OnDeinit |
| REASON_TEMPLATE | 应用新模板 | UninitializeReason, OnDeinit |
| SATURDAY | 星期六 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| SEEK_CUR | 文件指针的当前位置 | FileSeek |
| SEEK_END | 文件末尾 | FileSeek |
| SEEK_SET | 文件起点 | FileSeek |
| SENKOUSPANA_LINE | 闪光跨度 | 指标线 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | A线 | |
| SENKOUSPANB_LINE | 闪光跨度B线 | 指标线 |
| SERIES_BARS_COUNT | 当前时刻交易品种周期字节价值 | SeriesInfoInteger |
| SERIES_FIRSTDATE | 当前时刻交易品种周期第一数据 | SeriesInfoInteger |
| SERIES_LASTBAR_DATE | 交易品种周期最后字节的开仓时间 | SeriesInfoInteger |
| SERIES_SERVER_FIRSTDATE | 忽略时间表的服务器上交易品种历史中第一个日期 | SeriesInfoInteger |
| SERIES_SYNCHRONIZED | 当前时刻交易品种/周期的同步数据 | SeriesInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SERIES_TERMINAL_FIRSTDATE | 忽略时间表的客户端上交易品种历史中第一个日期 | SeriesInfoInteger |
| SHORT_MAX | 最大值，可以代表短型 | 数值类型常量 |
| SHORT_MIN | 最小值，可以代表短型 | 数值类型常量 |
| SIGNAL_BASE_AUTHOR_LOGIN | 作者登录 | SignalBaseGetString |
| SIGNAL_BASE_BALANCE | 账户结余 | SignalBaseGetDouble |
| SIGNAL_BASE_BROKER | 交易商名称（公司名） | SignalBaseGetString |
| SIGNAL_BASE_BROKER_SERVER | 交易商服务器 | SignalBaseGetString |
| SIGNAL_BASE_CURRENCY | Signal base currency | SignalBaseGetString |
| SIGNAL_BASE_DATE_PUBLISHED | 发布日期(提供订阅的日期) | SignalBaseGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SIGNAL_BASE_DATE_STARTED | 监控起始日期 | SignalBaseGetInteger |
| SIGNAL_BASE_EQUITY | 账户净值 | SignalBaseGetDouble |
| SIGNAL_BASE_GAIN | 账户所得 | SignalBaseGetDouble |
| SIGNAL_BASE_ID | 信号ID | SignalBaseGetInteger |
| SIGNAL_BASE_LEVERAGE | 账户杠杆 | SignalBaseGetInteger |
| SIGNAL_BASE_MAX_DRAWDOWN | 账户最大跌幅 | SignalBaseGetDouble |
| SIGNAL_BASE_NAME | 信号名称 | SignalBaseGetString |
| SIGNAL_BASE_PIPS | 利润点 | SignalBaseGetInteger |
| SIGNAL_BASE_PRICE | 信号订阅价格 | SignalBaseGetDouble |
| SIGNAL_BASE_RATING | 持仓评级 | SignalBaseGetInteger |
| SIGNAL_BASE_ROI | 投资利润率(%) | SignalBaseGetDouble |
| SIGNAL_BASE_SUBSCRIBERS | 订阅人数量 | SignalBaseGetInteger |
| SIGNAL_BASE_TRADE_MODE | 账户类型(0-真实，1-模拟，2-竞赛) | SignalBaseGetInteger |
| SIGNAL_BASE_TRADES | 交易数量 | SignalBaseGetInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SIGNAL_INFO_CONFIRMATIONS_DISABLED | 确认禁用标识 | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_COPY_SLTP | 复制止损和获利标识 | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_DEPOSIT_PERCENT | 入金比例(%) | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_EQUITY_LIMIT | 净值限制 | SignalInfoGetDouble, SignalInfoSetDouble |
| SIGNAL_INFO_ID | 信号id, r/o | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_NAME | 信号名称, r/o | SignalInfoGetString |
| SIGNAL_INFO_SLIPPAGE | 滑移(用于下市价单时同步持仓和复制交易) | SignalInfoGetDouble, SignalInfoSetDouble |
| SIGNAL_INFO_SUBSCRIPTION_ENABLED | 订阅启用标识 | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_TERMS_AGREE | "同意信号服务使用条款"标识, r/o | SignalInfoGetInteger, SignalInfoSetInteger |
| SIGNAL_INFO_VOLUME_PERCENT | 最大的使用入金百 | SignalInfoGetDouble, SignalInfoSetDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 分比(%), r/o | |
| SIGNAL_LINE | 信号线 | 指标线 |
| STAT_BALANCE_DD | 以货币计算，最大的结余跌幅。在交易处理过程中，结余可能有大幅度下跌，这里采用最大的值 | TesterStatistics |
| STAT_BALANCE_DD_RELATIVE | 以货币计算结余下跌，记录于百分比计算的最大结余跌幅时期(STAT_BALANCE_DDREL_PERCENT)。 | TesterStatistics |
| STAT_BALANCE_DDREL_PERCENT | 以百分比计算 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 最大结余跌幅。在交易处理过程中，结余可能有大幅度下跌，每次相关的下跌值都以百分比计算返回最大的值 | |
| STAT_BALANCEDD_PERCENT | 以百分比计算结余下跌，记录于货币计算的最大结余跌幅的时期(STAT_BALANCE_DD)。 | TesterStatistics |
| STAT_BALANCEMIN | 结余的最小值 | TesterStatistics |
| STAT_CONLOSSMAX | 一组亏损交易 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 中的最大亏损。该值小于或等于0 | |
| STAT_CONLOSSMAX_TRADES | 已经形成STAT_CONLOSSMAX（一组亏损交易中的最大亏损）的交易的数量 | TesterStatistics |
| STAT_CONPROFITMAX | 一组获利交易中的最大收益。该值大于或等于0 | TesterStatistics |
| STAT_CONPROFITMAX_TRADES | 已经形成STAT_CONPROFITMAX（一组获利交易中的最大收益)的交易的数量 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| STAT_CUSTOM_ONTESTER | 计算的自定义优化标准的值通过OnTester()函数返回unction | TesterStatistics |
| STAT_DEALS | 成交数量 | TesterStatistics |
| STAT_EQUITY_DD | 以货币计算，最大的净值跌幅。在交易处理过程中，净值上可能出现大幅度下跌，这里采用最大的值 | TesterStatistics |
| STAT_EQUITY_DD_RELATIVE | 以货币计算净值下跌，记录于百分比计算的最大净值跌幅 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 时期(STAT_EQUITY_DDREL_PERCENT)。 | |
| STAT_EQUITY_DDREL_PERCENT | 以百分比计算最大净值跌幅。在交易处理过程中，净值可能有大幅度下跌，每次相关的下跌值都以百分比计算返回最大的值 | TesterStatistics |
| STAT_EQUITYDD_PERCENT | 以百分比计算下跌，记录于货币计算的最大净值跌幅的时期(STAT | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | _EQUITY_DD)。 | |
| STAT_EQUITYMIN | 最小净值 | TesterStatistics |
| STAT_EXPECTED_PAYOFF | 期望收益 | TesterStatistics |
| STAT_GROSS_LOSS | 总亏损，所有负值交易的总和。该值小于或等于0 | TesterStatistics |
| STAT_GROSS_PROFIT | 总利润，所有获利（正值）交易的总和。该值大于或等于0 | TesterStatistics |
| STAT_INITIAL_DEPOSIT | 初始存款的值 | TesterStatistics |
| STAT_LONG_TRADES | 买入交易 | TesterStatistics |
| STAT_LOSS_TRADES | 亏损交易 | TesterStatistics |
| STAT_LOSSTRADES_AVGCON | 一组亏损交易的平均长度 | TesterStatistics |
| STAT_MAX_CONLOSS_TRADES | 最长的一 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 组亏损交易中的交易数量 STAT_MAX_CONLOSSES | |
| STAT_MAX_CONLOSSES | 最长一组亏损交易的总亏损 | TesterStatistics |
| STAT_MAX_CONPROFIT_TRADES | 最长的一组获利交易中的交易数量 STAT_MAX_CONWINS | TesterStatistics |
| STAT_MAX_CONWINS | 最长的一组获利交易的总利润 | TesterStatistics |
| STAT_MAX_LOSSTRADE | 最大亏损 - 所有亏损交易中的最低值。该值小于或等于0 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| STAT_MAX_PROFITTRADE | 最大收益 - 所有获利交易中的最大值。该值大于或等于0 | TesterStatistics |
| STAT_MIN_MARGINLEVEL | 保证金水平的最小值 | TesterStatistics |
| STAT_PROFIT | 测试后的净利润，STAT_GROSS_PROFIT 和 STAT_GROSS_LOSS 的总和 (STAT_GROSS_LOSS 始终小于或等于0) | TesterStatistics |
| STAT_PROFIT_FACTOR | 利润因子，等于 STAT_GROSS_PROFIT/STAT_GROSS_LOSS比率。如 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 果STAT_GROSS_LOSS=0，利润因子就等于DBL_MAX | |
| STAT_PROFIT_LONGTRADES | 获利的买入交易 | TesterStatistics |
| STAT_PROFIT_SHORTTRADES | 获利的卖出交易 | TesterStatistics |
| STAT_PROFIT_TRADES | 获利交易 | TesterStatistics |
| STAT_PROFITTRADES_AVGCON | 一组获利交易的平均长度 | TesterStatistics |
| STAT_RECOVERY_FACTOR | 采收率，等于STAT_PROFIT/STAT_BALANCE_DD的比率 | TesterStatistics |
| STAT_SHARPE_RATIO | 夏普比率 | TesterStatistics |
| STAT_SHORT_TRADES | 卖出交易 | TesterStatistics |
| STAT_TRADES | 交易的数量 | TesterStatistics |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| STAT_WITHDRAWAL | 从账户取款 | TesterStatistics |
| STO_CLOSECLOSE | 基于开盘价/收盘价的计算 | 价格常数 |
| STO_LOWHIGH | 基于最低价/最高价的计算 | 价格常数 |
| STYLE_DASH | 折线 | 绘画风格 |
| STYLE_DASHDOT | 折点线 | 绘画风格 |
| STYLE_DASHDOTDOT | 双折点线 | 绘画风格 |
| STYLE_DOT | 虚线 | 绘画风格 |
| STYLE_SOLID | 实线 | 绘画风格 |
| SUNDAY | 星期日 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| SYMBOL_ASK | 卖价—最佳买入信息 | SymbolInfoDouble |
| SYMBOL_ASKHIGH | 一天中最高买价 | SymbolInfoDouble |
| SYMBOL_ASKLOW | 一天中最低买价 | SymbolInfoDouble |
| SYMBOL_BANK | 当前报价支线 | SymbolInfoString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_BASIS | The underlying asset of a derivative | SymbolInfoString |
| SYMBOL_BID | 买价—最佳卖出信息 | SymbolInfoDouble |
| SYMBOL_BIDHIGH | 一天中最高买价 | SymbolInfoDouble |
| SYMBOL_BIDLOW | 一天中最低买价 | SymbolInfoDouble |
| SYMBOL_CALC_MODE_CFD | CFD方式—计算保证金和利润 | SymbolInfoInteger |
| SYMBOL_CALC_MODE_CFDINDEX | CFD指数模式—通过指数计算保证金和利润 | SymbolInfoInteger |
| SYMBOL_CALC_MODE_CFDLEVERAGE | CFD杠杆模式—通过杠杆贸易计算保证金和利润 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_CALC_MODE_EXCH_FUTURES | 期货模式 - 计算股票交易中，交易期货合约的保证金和赢利 | SymbolInfoInteger |
| SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS | 期货期权模式 - 计算期权交易中，交易期货合约的保证金和赢利。根据以下规则，保证金可能会因 Margin Discount 偏差的数额而减少：1. 如果买入持仓的价格（买入订单）少于预估 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 价格，MarginDiscount = Lots*((PriceSettle-PriceOrder)*TickPrice/TickSize)<br>2. 如果卖出持仓的价格（卖出订单）超过预估价格，MarginDiscount = Lots*((PriceOrder-PriceSettle)*TickPrice/TickSize)<br>在这里：<br>○ PriceSettle - 前一交易日的 | |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 预估价格（清算）；<br>○ PriceOrder - 订单（请求）中设置的加权平均持仓价格或持仓价格；<br>○ TickPrice - 订单价（ | |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 成本价格的点变化）<br>○ TickSize - 订单大小（最低价格变化的步骤） | |
| SYMBOL_CALC_MODE_EXCH_STOCKS | 交换模式 - 计算股票交易中，交易券的保证金和利赢 | SymbolInfoInteger |
| SYMBOL_CALC_MODE_FOREX | 外汇方式 — 计算利润和保证金 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_CALC_MODE_FUTURES | 期货方式—计算保证金和利润 | SymbolInfoInteger |
| SYMBOL_CALC_MODE_SERV_COLLATERAL | 担保模式 - 在交易账户中使用交易品种作为非交易资产。持仓的市值根据交易量，当前市价，合约大小和流动比率进行计算。价值包括在资产中，被加入到净值。这种交易品种的持仓提高了可用预付款的数额， | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 并用作可交易工具持仓的额外预付款（担保）。 | |
| SYMBOL_CURRENCY_BASE | 交易品种基础货币 | SymbolInfoString |
| SYMBOL_CURRENCY_MARGIN | 保证金货币 | SymbolInfoString |
| SYMBOL_CURRENCY_PROFIT | 利润货币 | SymbolInfoString |
| SYMBOL_DESCRIPTION | 交易品种描述 | SymbolInfoString |
| SYMBOL_DIGITS | 小数点后数字 | SymbolInfoInteger |
| SYMBOL_EXPIRATION_DAY | 当天结束前，该命令是有效的 | SymbolInfoInteger |
| SYMBOL_EXPIRATION_GTC | 该命令在无限时间周期中是有效的，直到它被明确删除 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_EXPIRATION_MODE | 到期模式允许命令标志 | SymbolInfoInteger |
| SYMBOL_EXPIRATION_SPECIFIED | 在命令中指定期限时间 | SymbolInfoInteger |
| SYMBOL_EXPIRATION_SPECIFIED_DAY | 有效期在订单中指定 | SymbolInfoInteger |
| SYMBOL_EXPIRATION_TIME | 交易品种交易结束日期（通常用于期货） | SymbolInfoInteger |
| SYMBOL_FILLING_FOK | 这个政策意味着交易只能执行指定的交易量。如果目前市场上无法提供必要的金融工具数量，那么订单将不会 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 执行。可以使用目前市场上可得到的几种提供来填写需要的交易量。 | |
| SYMBOL_FILLING_IOC | 在这种情况下，交易人同意以订单中指定的市场上最大可提供交易量，执行交易。如果订单不能完全填写，订单的可用交易量将被填写，余下的交易量将被取消。是否使 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 用IOC订单由交易服务器决定。。 | |
| SYMBOL_FILLING_MODE | 填充模式允许命令标志 | SymbolInfoInteger |
| SYMBOL_ISIN | ISIN系统中交易品种的名称（国际证券识别号码）。国际证券识别号码是一个12位数字字母代码，是证券的唯一识别码。交易品种的属性是由交易服务器方面决定的。 | SymbolInfoString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_LAST | 最后订单价格 | SymbolInfoDouble |
| SYMBOL_LASTHIGH | 一天中最高 | SymbolInfoDouble |
| SYMBOL_LASTLOW | 一天中最低 | SymbolInfoDouble |
| SYMBOL_MARGIN_INITIAL | 原始保证金表示每一笔保证金开仓成交量的数量 | SymbolInfoDouble |
| SYMBOL_MARGIN_MAINTENANCE | 维持保证金，在交易品种中建立保证金数量，从一笔订单中获得保证金利润，当客户账户变改时，使用客户资产检测系统，如果维 | SymbolInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 持保证金等于，使用原始保证金 | |
| SYMBOL_OPTION_MODE | Option type | SymbolInfoInteger |
| SYMBOL_OPTION_MODE_AMERICAN | 美式期权可以在超期前的任何交易日执行。可以为其指定买家可以执行该选项的周期 | SymbolInfoInteger |
| SYMBOL_OPTION_MODE_EUROPEAN | 欧式期权只能在指定日期执行（超七日，执行日，交付日） | SymbolInfoInteger |
| SYMBOL_OPTION_RIGHT | Option right (Call/Put) | SymbolInfoInteger |
| SYMBOL_OPTION_RIGHT_CALL | 买入期权给予 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 您在指定价格购买资产的权限 | |
| SYMBOL_OPTION_RIGHT_PUT | 卖出期权给予您在指定价格出售资产的权限 | SymbolInfoInteger |
| SYMBOL_OPTION_STRIKE | 期权的执行价格。期权买家可以购买（买入期权）或出售（卖出期权）标的资产的价格，期权卖家有义务出售或买入相应数量的标的资产。 | SymbolInfoDouble |
| SYMBOL_ORDER_LIMIT | 限价单被允许 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 买入限价和卖出限价） | |
| SYMBOL_ORDER_MARKET | 市价单被允许（买入和卖出） | SymbolInfoInteger |
| SYMBOL_ORDER_MODE | 允许的标识 订单类型 | SymbolInfoInteger |
| SYMBOL_ORDER_SL | 止损被允许 | SymbolInfoInteger |
| SYMBOL_ORDER_STOP | 止损订单被允许（买入止损和卖出止损） | SymbolInfoInteger |
| SYMBOL_ORDER_STOP_LIMIT | 止损限价订单被允许（买入止损限价和卖出止损限价） | SymbolInfoInteger |
| SYMBOL_ORDER_TP | 获利被允许 | SymbolInfoInteger |
| SYMBOL_PATH | 交易品种 | SymbolInfoString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 树形通路 | |
| SYMBOL_POINT | 交易品种点值 | SymbolInfoDouble |
| SYMBOL_SELECT | 在市场报价中选择交易品种 | SymbolInfoInteger |
| SYMBOL_SESSION_AW | 当前加权平均价格 | SymbolInfoDouble |
| SYMBOL_SESSION_BUY_ORDERS | 当时的买入订单数量 | SymbolInfoInteger |
| SYMBOL_SESSION_BUY_ORDERS_VOLUME | 买入订单的当前交易量 | SymbolInfoDouble |
| SYMBOL_SESSION_CLOSE | 当前平仓价格 | SymbolInfoDouble |
| SYMBOL_SESSION_DEALS | 当前时期的交易数量 | SymbolInfoInteger |
| SYMBOL_SESSION_INTEREST | 总持仓利息 | SymbolInfoDouble |
| SYMBOL_SESSION_OPEN | 当前持仓价格 | SymbolInfoDouble |
| SYMBOL_SESSION_PRICE_LIMIT_MAX | 当前最高价格 | SymbolInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_SESSION_PRICE_LIMIT_MIN | 当前最低价格 | SymbolInfoDouble |
| SYMBOL_SESSION_PRICE_SETTLEMENT | 当前结算价格 | SymbolInfoDouble |
| SYMBOL_SESSION_SELL_ORDERS | 当时的卖出订单数量 | SymbolInfoInteger |
| SYMBOL_SESSION_SELL_ORDERS_VOLUME | 卖出订单的当前交易量 | SymbolInfoDouble |
| SYMBOL_SESSION_TURNOVER | 当前的总流通量 | SymbolInfoDouble |
| SYMBOL_SESSION_VOLUME | 当前交易的总交易量 | SymbolInfoDouble |
| SYMBOL_SPREAD | 相关传播值 | SymbolInfoInteger |
| SYMBOL_SPREAD_FLOAT | 浮点传播迹象 | SymbolInfoInteger |
| SYMBOL_START_TIME | 交易品种交易开始日期（通常用于期货） | SymbolInfoInteger |
| SYMBOL_SWAP_LONG | 最低交易值 | SymbolInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_SWAP_MODE | 交易计算模式 | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT | 以客户入金货币的钱数收取利息 | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_CURRENCY_MARGIN | 以交易品种保证金货币的钱数收取利息 | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_CURRENCY_SYMBOL | 以交易品种基础货币的钱数收取利息 | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_DISABLED | 禁用利息（无利息） | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_INTEREST_CURRENT | 利息作为指定的年利息，从计算利息的工具价格收取（标准银行年是 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 360天） | |
| SYMBOL_SWAP_MODE_INTEREST_OPEN | 利息作为指定的年利息，从持仓的开盘价格收取（标准银行年是360天） | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_POINTS | 以点数收取利息 | SymbolInfoInteger |
| SYMBOL_SWAP_MODE_REOPEN_BID | 重新持仓收取利息。交易日结束，平仓。第二天以当前卖价 +/- 指定点数（参数SYMBOL_SWAP_LONG 和SYMBOL_SWAP_SHORT）重新持仓 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_SWAP_MODE_REOPEN_CURRENT | 重新持仓收取利息。交易日结束，平仓。第二天以收盘价+/- 指定点数（参数SYMBOL_SWAP_LONG 和SYMBOL_SWAP_SHORT)重新持仓 | SymbolInfoInteger |
| SYMBOL_SWAP_ROLLOVER3DAYS | 日翻滚收费 | SymbolInfoInteger |
| SYMBOL_SWAP_SHORT | 最小交易值 | SymbolInfoDouble |
| SYMBOL_TICKS_BOOKDEPTH | 显示在Depth of Market要求中的最大数量，交易品种无队列要求，值是0 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_TIME | 最后报价时间 | SymbolInfoInteger |
| SYMBOL_TRADE_CALC_MODE | 合约价格计算方式 | SymbolInfoInteger |
| SYMBOL_TRADE_CONTRACT_SIZE | 交易贸易合同 | SymbolInfoDouble |
| SYMBOL_TRADE_EXECUTION_EXCHANGE | 交换执行 | SymbolInfoInteger |
| SYMBOL_TRADE_EXECUTION_INSTANT | 立即执行 | SymbolInfoInteger |
| SYMBOL_TRADE_EXECUTION_MARKET | 交易执行 | SymbolInfoInteger |
| SYMBOL_TRADE_EXECUTION_REQUEST | 应邀执行 | SymbolInfoInteger |
| SYMBOL_TRADE_EXEMODE | 订单执行方式 | SymbolInfoInteger |
| SYMBOL_TRADE_FREEZE_LEVEL | 凝结交易操作的距离 | SymbolInfoInteger |
| SYMBOL_TRADE_MODE | 订单执行类型 | SymbolInfoInteger |
| SYMBOL_TRADE_MODE_CLOSEONLY | 平仓操作允许 | SymbolInfoInteger |
| SYMBOL_TRADE_MODE_DISABLED | 交易品种交易失去功效 | SymbolInfoInteger |
| SYMBOL_TRADE_MODE_FULL | 没有交易限制 | SymbolInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_TRADE_MODE_LONGONLY | 长仓允许 | SymbolInfoInteger |
| SYMBOL_TRADE_MODE_SHORTONLY | 短仓允许 | SymbolInfoInteger |
| SYMBOL_TRADE_STOPS_LEVEL | 止蚀盘当前收盘价格的最小空间 | SymbolInfoInteger |
| SYMBOL_TRADE_TICK_SIZE | 最小价格改变 | SymbolInfoDouble |
| SYMBOL_TRADE_TICK_VALUE | SYMBOL_TRADE_TICK_VALUE_PROFIT的值 | SymbolInfoDouble |
| SYMBOL_TRADE_TICK_VALUE_LOSS | 为亏损空间计算最小价位 | SymbolInfoDouble |
| SYMBOL_TRADE_TICK_VALUE_PROFIT | 为利润空间计算最小价位 | SymbolInfoDouble |
| SYMBOL_VOLUME | 最后订单成交量 | SymbolInfoInteger |
| SYMBOL_VOLUME_LIMIT | 交易品种单向（单买或单卖）持仓和 | SymbolInfoDouble |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 挂单可允许的最大总容量。例如，5手限定范围内，您可以拥有5手买入持仓交易量和下单5手卖出限价挂单交易量。但是在这种情况下，你不能下单买入限价挂单（因为单向总交易量将会超出限定范围）或下单超过5手的卖出限价交易量。 | |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| SYMBOL_VOLUME_MAX | 一笔订单中的最大交易量 | SymbolInfoDouble |
| SYMBOL_VOLUME_MIN | 一笔订单中的最小交易量 | SymbolInfoDouble |
| SYMBOL_VOLUME_STEP | 交易执行缓步的最小成交量 | SymbolInfoDouble |
| SYMBOL_VOLUMEHIGH | 当天最大订单 | SymbolInfoInteger |
| SYMBOL_VOLUMELOW | 当天最小订单 | SymbolInfoInteger |
| TENKANSEN_LINE | 转换线 | 指标线 |
| TERMINAL_BUILD | 客户端构造编号 | TerminalInfoInteger |
| TERMINAL_CODEPAGE | 在客户端建立的语言代码页数字 | TerminalInfoInteger |
| TERMINAL_COMMONDATA_PATH | 电脑中所有程序端的普通路径 | TerminalInfoString |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| TERMINAL_COMMUNITY_ACCOUNT | 标识表示程序端中存在MQL5.community授权数据 | TerminalInfoInteger |
| TERMINAL_COMMUNITY_BALANCE | MQL5.community的结余 | TerminalInfoDouble |
| TERMINAL_COMMUNITY_CONNECTION | 连接MQL5.community | TerminalInfoInteger |
| TERMINAL_COMPANY | 公司名称 | TerminalInfoString |
| TERMINAL_CONNECTED | 连接的交易服务器 | TerminalInfoInteger |
| TERMINAL_CPU_CORES | 系统的CPU内核数量 | TerminalInfoInteger |
| TERMINAL_DATA_PATH | 程序端数据文件夹存储 | TerminalInfoString |
| TERMINAL_DISK_SPACE | 程序端（代理）MQL5\Files文件夹的空闲磁盘 | TerminalInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 空间，MB | |
| TERMINAL_DLLS_ALLOWED | 使用DLL许可 | TerminalInfoInteger |
| TERMINAL_EMAIL_ENABLED | 在制定终端允许使用SMTP-server和login发送邮件 | TerminalInfoInteger |
| TERMINAL_FTP_ENABLED | 在制定终端允许使用FTP-server和login发送报告 | TerminalInfoInteger |
| TERMINAL_LANGUAGE | 程序端语言 | TerminalInfoString |
| TERMINAL_MAXBARS | 图表中的最大字节 | TerminalInfoInteger |
| TERMINAL_MEMORY_AVAILABLE | 程序端（代理）进程的空闲内存，MB | TerminalInfoInteger |
| TERMINAL_MEMORY_PHYSICAL | 系统的物理内 | TerminalInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 存，MB | |
| TERMINAL_MEMORY_TOTAL | 程序端（代理）进程的可用内存，MB | TerminalInfoInteger |
| TERMINAL_MEMORY_USED | 程序端（代理）使用的内存，MB | TerminalInfoInteger |
| TERMINAL_MQID | 标识表示存在MetaQuotes ID数据推送通知 | TerminalInfoInteger |
| TERMINAL_NAME | 程序端名称 | TerminalInfoString |
| TERMINAL_NOTIFICATIONS_ENABLED | 允许向智能手机发送通知 | TerminalInfoInteger |
| TERMINAL_OPENCL_SUPPORT | OpenCL支持的版本格式0x00010002 = 1.2。"0"表示OpenC | TerminalInfoInteger |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | L 不被支持 | |
| TERMINAL_PATH | 程序端文件夹启动 | TerminalInfoString |
| TERMINAL_PING_LAST | 最后知道的交易服务器的微秒ping值。一秒包含一百万微秒 | TerminalInfoInteger |
| TERMINAL_SCREEN_DPI | 屏幕上显示信息的分辨率是以每英寸一行的点数计算的（DPI） | TerminalInfoInteger |
| TERMINAL_TRADE_ALLOWED | 允许交易 | TerminalInfoInteger |
| TERMINAL_X64 | "64-位程序端" | TerminalInfoInteger |
| THURSDAY | 星期四 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| TRADE_ACTION_DEAL | 为规定参数的立即执行 | MqlTradeRequest |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 放置交易命令（市场命令） | |
| TRADE_ACTION_MODIFY | 修改先前放置的命令参量 | MqlTradeRequest |
| TRADE_ACTION_PENDING | 在制定环境下执行放置交易命令（待办订单） | MqlTradeRequest |
| TRADE_ACTION_REMOVE | 删除先前放置的待办订单命令 | MqlTradeRequest |
| TRADE_ACTION_SLTP | 修改折仓并取走开仓利润值 | MqlTradeRequest |
| TRADE_RETCODE_CANCEL | 交易者取消请求 | MqlTradeResult |
| TRADE_RETCODE_CLIENT_DISABLES_AT | 客户端无效自动交易 | MqlTradeResult |
| TRADE_RETCODE_CONNECTION | 与服务器 | MqlTradeResult |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 无连接 | |
| TRADE_RETCODE_DONE | 完成要求 | MqlTradeResult |
| TRADE_RETCODE_DONE_PARTIAL | 请求部分完成 | MqlTradeResult |
| TRADE_RETCODE_ERROR | 请求处理错误 | MqlTradeResult |
| TRADE_RETCODE_FROZEN | 命令或安置冻结 | MqlTradeResult |
| TRADE_RETCODE_INVALID | 无效请求 | MqlTradeResult |
| TRADE_RETCODE_INVALID_EXPIRATION | 请求中的无效命令截止日期 | MqlTradeResult |
| TRADE_RETCODE_INVALID_FILL | 无效命令填满字节 | MqlTradeResult |
| TRADE_RETCODE_INVALID_ORDER | 错误或禁止的订单类型 | MqlTradeResult |
| TRADE_RETCODE_INVALID_PRICE | 请求中的无效价格 | MqlTradeResult |
| TRADE_RETCODE_INVALID_STOPS | 请求中的无效访问 | MqlTradeResult |
| TRADE_RETCODE_INVALID_VOLUME | 请求中无 | MqlTradeResult |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 效成交量 | |
| TRADE_RETCODE_LIMIT_ORDERS | 待办订单数量达到限制 | MqlTradeResult |
| TRADE_RETCODE_LIMIT_VOLUME | 订单成交量和交易品种位置达到限制 | MqlTradeResult |
| TRADE_RETCODE_LOCKED | 处理锁住请求 | MqlTradeResult |
| TRADE_RETCODE_MARKET_CLOSED | 收市 | MqlTradeResult |
| TRADE_RETCODE_NO_CHANGES | 不改变请求 | MqlTradeResult |
| TRADE_RETCODE_NO_MONEY | 没有足够的钱实现请求 | MqlTradeResult |
| TRADE_RETCODE_ONLY_REAL | 只在流水账允许操作 | MqlTradeResult |
| TRADE_RETCODE_ORDER_CHANGED | 改变命令状态 | MqlTradeResult |
| TRADE_RETCODE_PLACED | 安置命令 | MqlTradeResult |
| TRADE_RETCODE_POSITION_CLOSED | 指定POSITION_IDENTIFIER的 | MqlTradeResult |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 持仓已经被关闭 | |
| TRADE_RETCODE_PRICE_CHANGED | 改变价格 | MqlTradeResult |
| TRADE_RETCODE_PRICE_OFF | 没有报价处理请求 | MqlTradeResult |
| TRADE_RETCODE_REJECT | 拒绝请求 | MqlTradeResult |
| TRADE_RETCODE_REQUOTE | 报价请求 | MqlTradeResult |
| TRADE_RETCODE_SERVER_DISABLES_AT | 服务器无效自动交易 | MqlTradeResult |
| TRADE_RETCODE_TIMEOUT | 超时取消请求 | MqlTradeResult |
| TRADE_RETCODE_TOO_MANY_REQUESTS | 太频繁的请求 | MqlTradeResult |
| TRADE_RETCODE_TRADE_DISABLED | 关闭交易 | MqlTradeResult |
| TRADE_TRANSACTION_DEAL_ADD | 添加交易到历史记录。订单执行执行操作或者执行账户结余操作。 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| TRADE_TRANSACTION_DEAL_DELETE | 删除历史记录的交易。可能会之前执行的交易从服务器删除的情况。例如，交易之前交易商移外部交易系统（交换）中已经被删除。 | MqlTradeTransaction |
| TRADE_TRANSACTION_DEAL_UPDATE | 更新历史记录交易。可能会有之前执行的交易变更到服务器上的情况。例如，交易在之前交易 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 商转移的外部交易系统（交换）中已经发生了变化。 | |
| TRADE_TRANSACTION_HISTORY_ADD | 添加订单到历史记录是执行或取消的结果。 | MqlTradeTransaction |
| TRADE_TRANSACTION_HISTORY_DELETE | 删除订单历史记录的订单。这个类型用于加强交易服务器的功能。 | MqlTradeTransaction |
| TRADE_TRANSACTION_HISTORY_UPDATE | 改变位于订单历史中的订单。这个类型用于加强交易服务 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 器的功能。 | |
| TRADE_TRANSACTION_ORDER_ADD | 新增新的持仓订单。 | MqlTradeTransaction |
| TRADE_TRANSACTION_ORDER_DELETE | 从持仓订单列表移除订单。设置相应的请求或执行（填充）和移动到历史记录后，订单可以从持仓订单中删除。 | MqlTradeTransaction |
| TRADE_TRANSACTION_ORDER_UPDATE | 更新持仓订单。更新不仅包括明显的客户端或交易服务器的变化还包括设置时的订单状态 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 的变化（例如，转换从ORDER_STATE_STARTED到ORDER_STATE_PLACED或从ORDER_STATE_PLACED到ORDER_STATE_PARTIAL等等）。 | |
| TRADE_TRANSACTION_POSITION | 改变与交易执行无关的持仓。这种事务类型表示持仓可以在交易服务器上改变。持仓交易量，开盘价，止损和获利 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 水平可以改变。更改的数据以MqlTradeTransaction结构通过OnTradeTransaction处理器来提交。持仓变化（添加，更改或关闭），作为交易执行的结果，不会引起TRADE_TRANSACTION_POSITION事务的出现。 | |
| TRADE_TRANSACTION_REQUEST | 服务器处理交易请求和处理结果的通知已 | MqlTradeTransaction |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| | 经收到。只有类型字段（交易事务类型）必须以MqlTradeTransaction结构分析该事务。OnTradeTransaction（请求和结果）的第二和第三参数必须分析额外的数据。 | |
| TUESDAY | 星期二 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| TYPE_BOOL | bool | MqlParam |
| TYPE_CHAR | char | MqlParam |
| TYPE_COLOR | color | MqlParam |
| TYPE_DATETIME | datetime | MqlParam |
| TYPE_DOUBLE | double | MqlParam |
| TYPE_FLOAT | float | MqlParam |
| TYPE_INT | int | MqlParam |
| TYPE_LONG | long | MqlParam |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| TYPE_SHORT | short | MqlParam |
| TYPE_STRING | string | MqlParam |
| TYPE_UCHAR | uchar | MqlParam |
| TYPE_UINT | uint | MqlParam |
| TYPE_ULONG | ulong | MqlParam |
| TYPE_USHORT | ushort | MqlParam |
| UCHAR_MAX | 最大值，可以代表无符字符型 | 数值类型常量 |
| UINT_MAX | 最大值，可以代表无符号整型 | 数值类型常量 |
| ULONG_MAX | 最大值，可以代表无符号长型 | 数值类型常量 |
| UPPER_BAND | 最高限制 | 指标线 |
| UPPER_HISTOGRAM | 上直方图 | 指标线 |
| UPPER_LINE | 上边线 | 指标线 |
| USHORT_MAX | 最大值，可以代表无符号短型 | 数值类型常量 |

© 2000-2025, MetaQuotes Ltd.

| 常量 | 描述 | 用法 |
| --- | --- | --- |
| VOLUME_REAL | 交易成交量 | 价格常数 |
| VOLUME_TICK | 赊欠成交量 | 价格常数 |
| WEDNESDAY | 星期三 | SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade |
| WHOLE_ARRAY | 代表很多条目剩下，直到数组末尾<br><br>如下，处理全部数组 | 其他常量 |
| WRONG_VALUE | 常量可以隐藏计算传递到任何 枚举 类型数组 | 其他常量 |

© 2000-2025, MetaQuotes Ltd.